An Intelligent tutoring system for the German language

Kenneth E. Staffan
Title of Thesis: An Intelligent Tutoring System for the German Language

I, Kenneth E. Staffan, prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

100 Indigo Creek Drive
Rochester, New York 14650

or electronically at:

staffan@serum.kodak.com

date 10-May-93
Acknowledgements

All the thanks in the world go to the following people: Kevin Donaghy and Hank Etlinger, for their advice and guidance throughout the project, my wife, Mary, for bearing an inordinate amount of the home and family burden while I worked, and for supplying the original Intelligent Tutoring System concept, and last, but by no means least, my young son, Ben, for being the bright spot at the end of each long day of work and school. May he develop a thirst for learning, himself.
Abstract

This thesis report describes the design and implementation of a prototype Intelligent Tutoring System (ITS), intended to assist students of the German language. Very early in the study of a foreign language, the student is faced with the difficulties of sentence construction. Not only are there numerous rules and combinations to deal with, but it is difficult to verify attempts when the teacher is unavailable. Individual words can be looked up in a dictionary, but the student must often rely on stumbling across a sentence of similar construction in order to verify a trial sentence.

A variety of Computer-Assisted Language Learning (CALL) tools have been developed. Many have been criticized for not being user friendly, containing material which does not match the course curriculum, being inflexible, or being just plain incorrect. The prototype system developed for this thesis experiments with several characteristics – an object-oriented design approach, a masking technique using dynamically built patterns to bridge the gap between hard-coded and full artificial intelligence approaches, and a C++ implementation. It attempts to draw on past failures, as well as past successes.

The system described here provides a means for practicing sentence construction, with interactive diagnosis and feedback. A phrase or sentence is presented to the student for translation. The response is then checked for correctness. If the answer is incorrect, the student is given the option of trying again, receiving increasingly more specific hints, or having the system display the response it was expecting.
Key Words and Phrases


Subject Codes

ACM Computing Review Codes

D.1.5 Software
  Programming Techniques
  Object-Oriented Programming

I.2.1 Computing Methodologies
  Artificial Intelligence
    Applications and Expert Systems

I.2.7 Natural Language Processing

K.3.1 Computing Milieux
  Computers and Education
    Computer Uses in Education

Inspec (IEEE/IEE) Computer and Control Abstracts Codes

1230 Artificial Intelligence
6170 Expert Systems
6180N Natural Language Processing
7110 Education
1. INTRODUCTION AND BACKGROUND

1.1. Problem Statement

When a foreign language student leaves the classroom environment and teacher, a valuable resource is lost. The student no longer has an expert to use as a resource for understanding or clarification. Text books and other "static" media are not always effective reference (or tutoring) tools. A dictionary, for instance, can be used if a student wishes to look up an individual word, but may be of little help if the question concerns sentence construction or other concepts. Intelligent Tutoring Systems (ITS), and other forms of Computer-Assisted Instruction (CAI), have attempted to extend the resources available to a student for practice and reference.

Computer-Assisted Instruction has been a fairly popular research area, in both the academic and commercial worlds, since the early 1960's. A variety of student- and teacher-oriented products are currently on the market or in experimental usage. Questions, however, about the lack of effectiveness of these products, combined with the apparent lack of real progress in the field, are what prompted one researcher to speculate on whether the field might better be called CIA – Computer-Inhibited Acquisition [Farrington, 1986]. Part of the difficulty has been in the cross-disciplinary nature of the task. Not just a computer or parsing exercise, CAI researchers must take into account accepted methods in the fields of education, psychology, and, of course, the subject which is to be taught.

This thesis project addresses one area in which an intelligent tutoring system can be of assistance – the area of phrase and sentence construction. This is an area where resources are scant, when no expert is accessible. To verify the correctness of a sentence, the student may have to rely on stumbling across one of similar structure in a text book. Even this may be inaccurate, as different vocabulary may require a variation in the structure of the sentence. Sentence construction is not an advanced topic, either. Most language courses begin introducing some sentence forms with the very first class. Already, the best CAI packages are challenged.

The ITS prototyped as part of this thesis experiments with a masking technique of diagnosis, using run-time generated patterns to identify the discrepancies in a student response. The thesis also experiments with the object-oriented analysis, design and implementation of such a system. A fully functional (though limited in knowledge depth) system has been designed and implemented to illustrate and exercise these concepts. The system presents phrases to the student for translation, and diagnoses the responses. Hints and additional chances are offered if the student desires.

1.2. Previous Work

In the early 1960's, there was not much distinction between Computer-Assisted Language Learning systems, and other disciplines of CAI. Existing systems were mainly
of the drill–and–practice variety, the equivalent of electronic flashcards. By the late 1960's, attempts were being made to use interactive feedback to control the systems' operation [Uhr,1969]. The earliest forms of feedback–control involved merely re–asking problems which were previously answered incorrectly. By the early 1970s, attempts were being made to adjust the difficulty of problems based on performance [Woods,1971]. Discussion ensued on when help should be offered to the student, and whether the student or the tutoring system should determine the level of difficulty of the problems [Hartley,1973].

By the 1980s, many projects were concentrating on the best way to represent the state of the student's knowledge, in order to optimally adapt to it [Goldstein,1981], [Burton,1981a–b], and [Sleeman,1981a–b]. Research had also diverged into two distinct areas, template software for teachers to author their own computer–assisted lessons, and artificial intelligence and auto–adaptation programs which supported little, if any, teacher modification [Boyd,1982].

Today, CALL research and CALL systems typically fall into two broad categories. The largest and most sophisticated systems are developed and maintained by large universities for internal use. They have evolved over many years, and are highly customized to support their own language curriculums. Likewise, the most sophisticated systems in related areas (for instance, business document critiquing systems), have been developed internally by large corporations and large universities. These types of organizations have the advantage of large computing resources, and projects which slowly evolve and grow as they reach the more subtle hurdles. Another distinct advantage is that, as their own customers, the researchers in these universities and corporations have a continuous open channel for feedback, both positive and negative. At the other end of the spectrum, most commercially available CALL software is much less sophisticated. The intended customers, typically individuals and pre–college educational institutions, tend to have much more limited computing resources, and only a very small budget for educational software. In order to meet these customer constraints, and to bring such products to market in a timely manner, most of these packages address very narrow topics at the beginner level.

One large non–commercial effort that has been in–progress since the early 1970's is at Concordia University (Canada). They began experimenting with both student– and teacher–oriented CALL programs, the main focus of which were to improve the effectiveness of their own remedial and second language English programs. Narrow topics, such as specific verb tenses, were addressed individually, with the hopes of eventually building a library of software for a large range of grammatical and syntactical topics. One of the more ambitious later projects was a "whole composition" analyzer, which attempted to diagnose student–constructed sentences [Boyd,1982]. Several notable concepts were illustrated in this project. First was the decision to check sentences for common forms of faults, rather than on a true AI analysis of the sentence for correct–
ness. It proved that a benefit could be still be realized, while narrowing the scope and the complexity of the task. Another successful concept was the extensive compiling of statistics, including comments solicited directly from the students at the end of lessons, for adjusting and enhancing the diagnostics (for continuous improvement).

In the early 1980's, a similar program was being tested at Kings College (Scotland). This program was used for instruction in French, and illustrated some other successful characteristics [Farrington, 1982]. A common complaint had been that CAI software often did not recognize alternate correct answers. This program specifically researched and hard-coded multiple correct answers to each problem. User-friendliness (to both those familiar with computers, and those not) was another frequent complaint about early CAI attempts. This program offered hints, encouragement, praise, etc., if desired by the student. This was an improvement over the "display a problem", "read student response", "if incorrect, print correct answer", "display next problem" cycle of earlier drill-and-practice software. Other attention was paid to user-friendliness, giving the student control over the level of help, and the ability to exit at any time. Similar to the Concordia University program, incorrect answers were stored so that future enhancements could handle new variations of answers and new types of errors.

During the same time period, CALL research was also being conducted at the University of Waikato (New Zealand). Various projects examined issues around natural language tutoring. The pros and cons of using the first language (meaning the student's normal language, as opposed to the one being taught) on the screen were explored. This issue is still not resolved, though. It was also concluded that exercises were more effective when not categorized in such a fashion that the student already knew what problem area was being tested. The projects also experimented with overall user-friendliness of CALL systems.

Several researchers have addressed alternatives to hard-coding answers to exercises. One possibility is to hard-code a variety or hierarchy of correct answers, as mentioned earlier. An alternative is to "can" common wrong answers as well as correct answers, for efficient trapping of errors [Ferney, 1989]. One successful program which uses canned knowledge, the LITTRE program, pushes the hard-coding technology to its limits [Farrington, 1986]. The program has proven very useful, and comes across as quite knowledgeable in French. The disadvantage has been that each individual sentence requires extensive analysis before being canned.

At IBM Research, the work of Roy J. Byrd, and associates, has contributed much useful information on word-based storage and morphological analysis [Byrd, 1983, 1986, 1988]. All of these works have been based on a component dictionary system called UDICT. Intended as a building block for general-purpose natural language processing, the concepts for knowledge representation are relevant to CALL. There are two components to the UDICT system. A database component consists of words, feature information (such as part-of-speech, gender, etc.), and morphological information (rules on
how the variations of the word are formed). The second component is a morphological analyzer to process the feature information in order to match variations to the base word. Most of the database information is encoded in binary fashion (e.g. the "masculine" feature would be either on or off) so as to take a minimum of space, and allow fast access. So far this representation has been undertaken for English and French. The straight-forward implementation, however, lends itself to relatively easy adaptation to other western languages.

An examination of the currently available commercial CAI packages for the German language provides some additional insight into the state of the art. Included, as Appendix D, is a list of currently available software packages for German, along with brief descriptions of each. Nearly all are variations of the drill-and-practice method (as is this thesis project). Approximately half of the packages handle single-word vocabulary only, while the other half attempt in some way to address sentence-level constructs. Approximately half have chosen to present the exercises in a game format. While all are described briefly in the Appendix, two which most directly address the same domain (sentence construction) as this thesis are discussed below.

The German Word Order program is an interesting variation on sentence construction exercises. All sentences are canned. The program chooses a sentence from its internal list, then prints the words in random order. The student must attempt to reconstruct the correct sentence. Because the student must use exactly the same words as the original sentence, parsing for correctness becomes much easier. One problem with this particular program, is that it does not accept correct sentences which do not match the original. To take an English example, "The truck hit the car" would not be accepted if the system's version of the original sentence was "The car hit the truck". This can be particularly restrictive if an alternative variation of the correct sentence retains the same meaning as the original. In this case, the student may be misled into believing that only the one form is correct.

The program gives interactive feedback while the student constructs the sentence. As each word is entered, the computer tells whether or not it is correct. This helps to alleviate the problem of alternative correct sentences (in the above example, "truck" would have been flagged as incorrect, giving the student the chance to try "car" instead). Hints are also available along the way. The student can review a brief lesson on the concept which is employed in the current sentence, or receive the correct answer, at any point.

Another program which deals with sentence construction is the Dasher program. It begins much like the system implemented for this thesis, by presenting a sentence for translation. The difference is that it does not attempt any intelligent diagnosis or tutoring. The student response is compared letter-to-letter with the expected response (also greatly simplifying correctness parsing), and echoed back to the student with dashes (hence the name) in place of any discrepancies. This occurs a specified number of times before the correct answer is given. The re-try count is configurable.
The field of CALL research is still drawing interest. One researcher comments that the current trick lies in discovering and exploiting the knowledge representations which would allow systems to take shortcuts, like a human would, in analyzing sentences [Cameron,1989]. Another researcher comments that the pioneering spirit is still alive, and that we are still building a base of background material to establish clear direction [Last,1989]. He also criticizes projects tackled by educators without enough programming theory, and projects tackled by programmers without regard to education theory. These are not unfounded complaints, and such needs of CAI/CALL researches have prompted education experts to publish materials intended for the programming audience [Fox,1989], [Dhaif,1990]. Another resource to CAI/CALL researchers are papers intended for teachers on how to select good CAI software [LaReau,1989]. Both contain information of value to CAI designers – advice on effective and ineffective teaching techniques, user interface techniques, and the perceptions of students and teachers.

1.3. Scope of Project

The editors of a collection of computer-assisted language learning research papers, commented in their introduction:

"As one reads through the chapters in this book, one realizes that much has been achieved and that much more will be achieved by selecting ideas which have been tried out by individuals and by amalgamating them." [Cameron,1986]

The primary deliverable for this thesis is a prototype Intelligent Tutoring System which attempts to do just that. It is accompanied by design, user, and administrative documentation. It combines some of the concepts described in the previous section, to provide a small package supporting semi-generated phrases, intelligent critiquing of student responses, assistance to the student, if desired, and information gathering for further improvement of its capabilities.

The semi-generated phrases mark a hybrid between the all-canned phrases of such programs as the LITTRE program, and true random sentence generation. The system is programmed to diagnose specific forms, or models, of phrases and sentences. These models draw from pools of semantically correct selections, to eliminate the problem of generating nonsensical phrases. However, all phrase construction, and diagnostic pattern construction, occurs at run-time. The details of diagnosing a specific phrase are abstracted so that they apply to the entire model. The run-time construction, then, allows the system to correctly diagnose whatever vocabulary happens to be used for a specific exercise. Besides the programming advantage, this also makes adding phrases or sentences, of existing forms, very easy.

Externally, the tutoring system is fairly simple. It handles a single type of exercise, sentence translation, with diagnostic feedback. When the student logs in, the system queries the student’s current knowledge level. All vocabulary and sentence models within
the system are marked with the chapter in the textbook in which they are introduced. By requesting the student's knowledge level, expressed as the highest chapter completed in the textbook, the system can present only sentence forms and vocabulary which it knows the student has been exposed to. The knowledge level concept is discussed in more detail in section 1.3.2. below, and in the Design and Implementation chapter later in the report. A few general user-interface rules apply. At any user prompt, the student is able to ask for help or exit the program. At any user prompt within a specific exercise, the student is able to abort that exercise and begin a new one. Help text includes information on what is expected at the specific prompt, as well as on these general rules of control. A typical single-exercise dialogue consists of:

  - The system presents a phrase or sentence to the student, for translation.
  - The system reads the student's response.
  - If the response is correct, the system says so, and a new phrase is presented for translation.
  - If the response is incorrect, the student is asked if he or she would like to try again, receive the correct answer, or receive a hint from the system.
  - If the student chooses to try again, the system reads another translation response (and attempts to diagnose it).
  - If the student requests the correct answer, it is given, and the system presents a new phrase or sentence for translation.
  - If the student requests a hint, the system attempts to provide a clue as to what is wrong with the response, without giving away the answer. If subsequent hints are requested, the system attempts to give successively more detailed clues.
  - During the course of the exercise, statistics and information about specific responses are logged, for the purpose of future enhancements to make the system more robust.

Additional detail about the processing and user interface can be found in the Software System Description chapter later in this report, and in the User and System Administration Manuals included as appendices. Because the project had multiple objectives, the scope of each is addressed below.

1.3.1. System Functionality

Exhaustively addressing all possible sentence constructions for a language would obviously have not been manageable as a thesis project. It was a goal, however, to implement enough variety to illustrate the viability of its concepts. This project does not constitute a major leap forward in CALL technology, but it is hoped that its experimental components represent small steps in the evolution towards more effective tools. By taking concepts from successful systems which used hard-coded data, and attempting to abstract the techniques, the prototype attempts a blending of canned and "intelligent" processing. Another key
1.3.2. Design and Implementation

The tutoring system is highly tied to a specific textbook. This allowed the phrase model and vocabulary selection, and choice of diagnostic errors, to be done in a logical and pre-established way. This also provided a convenient means for defining the "knowledge level" concept. Using "highest chapter completed" as a measure of knowledge level provides a clear way to mark the internal knowledge. Phrase models and vocabulary are tagged with the chapter in which they were introduced. This also keeps the knowledge level concept straightforward for the student user. The use of the knowledge level addresses a comment complaint against CAI software – that the material does not align with different student's capabilities.
As mentioned, an object-oriented approach was taken. The scope of this objective was to apply object-oriented analysis and design techniques to the requirements for the proposed system, and then implement the design using a language which supports object-oriented constructs. Job-Oriented Object Analysis (JOOA) [Nichols, 1991] and Class-Responsibility-Collaboration Design (CRC) [Wirfs-Brock, 1990] methodologies were used. The analysis, design and implementation details are documented in Appendix A. The tutoring system runs on the Computer Science Department Sun network.

1.3.3. System Internals

The principle objective, from an architecture standpoint, was the implementation of the pattern masking diagnosis technique. Most of the software development was related to integrating and illustrating the feasibility of this approach.

The dictionary portion of the system is very similar to the scheme described by Byrd, et al. Words and feature information are stored in the dictionary. When the system is building phrases to present to the user, it requests the variation of words that it needs by specifying the feature information. The dictionary is also used when diagnosing the student's response, to determine if incorrect words are variations of the expected word, synonyms, etc. The dictionary is limited to the vocabulary that the system currently supports (based on knowledge level). A possible future enhancement, to use a full external dictionary, is discussed in the conclusions.
2. SOFTWARE SYSTEM DESCRIPTION

2.1. Introduction

A later Design and Implementation chapter represents the dense, but thorough, documentation of the system development and architecture, using the techniques and terminology described in [Nichols, 1991a–b] and [Wirfs–Brock, 1990]. What follows in this chapter is the "plain language" system description, intended as an overview of both its use and the underlying design. Two other documents of interest exist, and have been included as appendices to this report. They are the User Manual, and the System Administration Manual (appendices A and B, respectively). The User Manual is aimed at the student user who wishes to use the tutoring system to practice translation. The System Administration Manual covers two main topics, the organization and building of source code, and the availability of data gathered by the system to characterize and improve its performance.

2.2. User Interface Overview

The prototype system's user interface is not terribly sophisticated. It is text–based, interactive I/O. The user types input at the keyboard, and the system prints output to the screen. Most of the user input consists of attempted phrase translations, and command numbers chosen from menus along the way. Additional detail can be found in the User Manual appendix and the Sample System Runs chapter later in this report, but the brief interchange below illustrates the look and feel of the system. In system logs throughout this report, text printed by the system will appear in bold, and text typed by the user will appear in italics.

```
sys27> glt

German Language Tutor, version 1.0

If you are not sure how to begin, type "?".

Please Login: ken

Enter highest chapter completed (currently 2)? 3

Exercises will be limited to chapter 3 and earlier.

You will now be asked to translate various phrases and sentences.

Translate:

   the table is round

to German:
```
Translate:

the table is round

to German:

*der Tisch ist rund*

Yes! That is the expected response.

In the example shown, the student invokes the tutor using the 'git' command, and then logs in using the username 'ken'. Valid usernames are assigned by the system administrator. Two types of users are supported, the "student" and the "system administrator". Upon logging in, student users enter the exercises, as illustrated above. System administrator users have different features available. Refer to the respective manuals for additional details on each.

The types of errors diagnosed are specific to different forms of phrases. Different levels of diagnosis are typically available. If the user above had requested another hint,
instead of choosing to try another translation, the system would have reported that it was the tense of the verb which was wrong. If the user had requested yet another hint, the system would have reported that the choice "war" was the imperfect form of sein, but that it was expecting the present tense. Eventually, the system would have reported that it could not give anymore clues without revealing the answer, and the option of receiving a hint would have disappeared from the menu.

On-line help is available anytime the system is waiting for input, and the student is always free to quit the current exercise, or exit the system altogether.

2.3. Internal Architecture Overview

The design and implementation of the system was done using object-oriented methods, and that terminology is used here for consistency. The Terminology sections of the Design and Implementation chapter may be referred to for additional clarification, if needed.

2.3.1. High-Level Description

The fundamental object, from a processing standpoint, is the PhraseModel. The PhraseModel represents everything that the system knows about a specific form of phrase or sentence – what semantically correct vocabulary can be used, how to build a syntactically correct phrase from this vocabulary, and how to diagnose a student’s translation attempt. The normal execution of the system is basically a sequence of exercises generated by choosing PhraseModels from the list which the system supports.

The fundamental object, from a functionality standpoint, is the TranslationTree. This is a binary tree of masking patterns, through which the student response falls as the system tries to diagnose it. An initial diagnosis is made at the first pattern which matches the response, and additional help (hints) can be supplied to the user by matching subsequent patterns. The TranslationTree and its nodes generate these patterns at run-time, so that they work with the vocabulary which was chosen for the specific exercise. The normal execution of a single exercise consists of the PhraseModel generating a phrase to be translated, and the TranslationTree processing student responses.

Another important object, as far as objectives of the system go, is the Log. This object gathers run-time data to be used in characterizing system usage, and improving its diagnostic capability.

The remainder of the objects in the system function primarily as support for those just mentioned. These are discussed further in the next section.

2.3.2. Detailed Description

A PhraseModel works for a particular form of phrase or sentence. This is represented by a PhraseDescriptor. The PhraseDescriptor contains the information necessary to create a syntactically correct phrase of the intended form. In order
to create a phrase to be presented to the student for translation, the PhraseModel
draws a semantically correct BasePhrase from a pool of choices. A BasePhrase
is an indication of valid vocabulary selections for this form of phrase, with no
syntactic information included. The PhraseDescriptor uses the BasePhrase,
and builds the syntactically and semantically correct phrase which the Phrase-
Model presents to the user.

The TranslationTree works from the same BasePhrase, and builds the system’s
expected response. All diagnosis is done by matching the student response
against patterns, and the expected response is essentially the first pattern in the
binary tree – a pattern with no wildcards. If the student response matches the
expected response, this exercise is complete, the system logs some data and sta-
tistics, and another PhraseModel is chosen.

If the student response does not match the expected response, then it is fed to
the first PatternNode in the binary tree. The pattern node builds a pattern, which
consists of the expected response with some strategically placed wildcard(s).
This pattern is matched against the student response, and if it fails, the student
response is fed down the left branch of the tree until a match is found. Each
generated pattern is different, and/or increasingly more generic, in an effort to
isolate the problem area.

Once the student response matches a pattern, the system reports that the re-
sponse was not the expected one, and asks the student how to proceed. If the
student wishes to try again, the system resets to the root of the TranslationTree,
and waits for another attempt. If the student asks for a hint, the one which is
stored at the current PatternNode will be printed. If the student asks for addi-
tional hints, the system will feed the student’s response down the right branch
of the tree. Subsequent patterns will try to further isolate or qualify the prob-
lem, and hints at those levels will be increasingly more specific. Once the tree-
parsing hits a leaf node, though, the system will tell the user that it can diagnose
no further.

A Dictionary object supports the system, by returning the derived forms of
words based on specified syntactic information. The PhraseModel and Transla-
tionTree use this as they build the presentation and expected phrases.

A Statistics object maintains run-time counts, such as how many exercises have
been attempted during the current session. This information is made available
to the user periodically, as well as logged for future inspection.

Further detail about these, and additional, objects, may be found in the Design
Document appendix. The next section is a system walk-through to illustrate
how these objects work with an example.

2.3.3. Walk-through to Illustrate Internal Function

This walk-through is included as a way to show how the different objects inter-
act, and how the various dynamic objects are created and used. It also shows
how the pattern-tree diagnosis concept works.
2.3.3.1. Static Data

Before actually beginning the processing, it is important to see what static objects are in place. This is the system's knowledge base. A BaseWord is a primitive data element in the system. It is not actually a word, but rather a key into the dictionary indicating a specific word, but not the word's form. For example the BaseWord for "man" would also be the base word for "men". As will be seen later, a WordDescriptor contains information about what form of a word is required, and a BaseWord and WordDescriptor are all that is needed to obtain a Word from the dictionary. BaseWords are combined to form BasePhrases, which is one of the objects that a PhraseModel contains. BasePhrases are grouped into BasePhraseLists, which in turn are grouped into BasePhrasePools. The need for two levels of grouping will be illustrated below. Each PhraseModel has its own BasePhrasePool to draw from. To begin the example, here are two BasePhraseLists, each containing two BasePhrases:

BPL-1: "the", "man", "to be", "good"
       "the", "dog", "to be", "hungry"

BPL-2: "a", "boy", "to be", "energetic"
       "a", "bird", "to be", "noisy"

As mentioned earlier, a PhraseModel has a PhraseDescriptor which specifies the form of phrase or sentence represented by this model. This contains syntactic information. The PhraseDescriptor also contains information on the order in which the phrase must be built. This is so that words which must agree with each other (in gender, number, etc.) will. Here is an example PhraseDescriptor:

PD-1: "article, gender-dynamic", "singular noun",
      "present tense verb", "adjective"

Fill-in order: 2, 1, 3, 4

The actual elements of the PhraseDescriptor are WordDescriptors. The WordDescriptors contain feature (or attribute) information describing the required form of the word. For example the noun specified above is singular, as opposed to plural. A special characteristic of WordDescriptors is that features may be marked as "dynamic" as is the gender of the article in the above example. This supports the run-time building of phrases where words must agree with each other, and will be shown later.

The following should illustrate the difference between BasePhrasePools and BasePhraseLists. A BasePhrase and a PhraseDescriptor together specify a syntactically and semantically correct phrase. The following example BasePhrasePool is composed from both of the example BasePhraseLists.

BPP-1: BPL-1
      BPL-2

A PhraseModel using the example PhraseDescriptor, PD-1, and the example BasePhrasePool, BPP-1, would be able to generate the following valid phrases:

The man is good.
The dog is hungry.
A boy is energetic.
A bird is noisy.

BPL–1 and BPL–2 are separate entities, though, because they will not always be valid for the same PhraseModel/PhraseDescriptors. As an example of this, suppose the fixed_phraseDescriptor for a different PhraseModel is:

PD–2:    "article, gender–dynamic", "plural noun", "present tense verb", "adjective"
          Fill–in order: 2, 1, 3, 4

A valid BasePhrasePool is:

BPP–2:    BPL–1

which would generate the valid phrases:

The men are good.
The dogs are hungry.

BPP–2 cannot contain BPL–1, however, because the generated phrases would not be valid:

A boys are energetic.
A birds are noisy.

BasePhraseLists, then, exist so that subsets of pools can be shared between PhraseModels.

The first example PhraseModel currently consists of BPP–1 and PD–1. The next component of a PhraseModel is its TranslationTree. A TranslationTree object is at the root of the binary pattern–matching tree. Like the PhraseModel, it also contains a PhraseDescriptor. This describes the expected response. In many cases, this will be the same PhraseDescriptor that the PhraseModel uses. In those cases were it is not the same, the TranslationTree will also contain a BasePhraseTranslator object which describes how to translate the original BasePhrase to the new language. This handles cases where the translated phrase does not take on the same form as the original (e.g. greater/fewer words or different word order). The example here does not require this, so no additional PhraseDescriptor is included.

The final static components of the PhraseModel and TranslationTree are the individual PatternNodes. The tree itself is pre–determined, but there are dynamic actions which occur during an exercise, and these will be illustrated below. In order to have an example to work from, consider the following tree (without yet specifying the details of each pattern):
That completes the description of what the system knows before starting an exercise. The next section examines the processing which would occur using this example data.

2.3.3.2. Run-time Processing

The first step of an exercise is choosing a BasePhrase to work with. It is intended that this be a random selection from the BasePhrasePool (BPP–1, in this case). For this example, say that "the" "man" "to be" "good" was chosen. The PhraseDescriptor (PD–1) is used to generate the phrase to be presented to the student. A special object, the DynamicWordDescriptor is used along the way to help resolve dynamic information in the WordDescriptors. As the phrase is constructed, feature information is set in the Dynamic WordDescriptor. Using the chosen BasePhrase, and following the fill-in order specified by the PhraseDescriptor, the second word of the presentation phrase would be obtained from the dictionary (for this example I’m assuming a German to English translation by the student, so the German forms of the word are filled in):

\[
\text{presentation} = \text{Mann} \ \text{_____}
\]

In addition to setting this word, its feature information is set in the DynamicWordDescriptor (specifically, the fact that the gender of this word is masculine). As each subsequent WordDescriptor is used, it is first checked to resolve any dynamic feature information against the DynamicWordDescriptor. The next WordDescriptor would then be resolved to "masculine article", and would be obtained from the dictionary:

\[
\text{presentation} = \text{der Mann} \ \text{_____}
\]

These steps are repeated until the entire presentation phrase has been constructed. It is then presented to the user:

Translate:

\[
\text{der Mann ist gut}
\]

to English:

At this point, the TranslationTree takes over, and waits for the student’s response. It builds the Phrase that it expects (in this case "the man is good") and compares the two. If the response is what the system expected, the system would inform the user, and this exercise would be over. For sake of example, though (and because gender is a convenient error to illustrate), consider the following exercise:
Translate:
the man is good
to German:

In this case, the correct response would be "der Mann ist gut". To illustrate error diagnosis, assume that the response was "das Mann ist gut" (gender of the article is incorrect).

Because the match with the expected response fails, the TranslationTree object feeds the actual response into the pattern tree. The following is simplified example of how a fragment of a pattern tree might look. Every PatternNode in the tree is similar in structure. Each contains the rules necessary for building the pattern needed at that level, and a hint as to the nature of the discrepancy at that level. The patterns are dynamically built, and the ones shown below indicate how they might look for this example vocabulary. (In the notation here, '*' represents the single-word wildcard.)

\[
\begin{align*}
P-3 &= "*\text{ Mann ist gut.}" \\
\text{Hint} &= \text{first word wrong} \\
P-6 &= "\text{der }*\text{ ist gut.}" \\
\text{Hint} &= \text{second word wrong} \\
P-7 &= "\text{das Mann ist gut.}" \\
\text{Hint} &= \text{gender wrong}
\end{align*}
\]

The student response would fall through the tree until it first matched a pattern. In this case, that would be P-3. At this point the system will ask the student what he or she wishes to do. As mentioned earlier, if the student chooses to try again, or to receive the correct answer, the system will leave the diagnosis tree. If the student requests a hint, though, the system will print the hint associated with the current PatternNode. In this case, it would tell the user that it did not like the first word in the response. If the user asks for another hint, the system will attempt a match down the right branch of the tree. In this case, a match is found with the pattern P-7, and the next hint is printed, this time telling the user that it was the gender of the first word that caused the problem. This can continue as long as the system has a right branch to follow. Once it hits the leaf, though, it will tell the user that no further diagnosis is possible. The left branch, P-6 would only have been taken if the response had not matched at P-3 originally. Internally, the left branch is considered the "no-match" path (taken while looking for the first match) and the right branch is considered the "match" path (taken to supply hints, as long as it continues to match the user’s response).

That concludes a single exercise processing walk-through. Diagnosis will continue until the user answers correctly or quits the current exercise, then a new exercise will begin.
3. SAMPLE SYSTEM RUNS

3.1. Introduction

The following sections consist of excerpts from logs of sample sessions with the system. They have been broken into selections which illustrate different features and interesting aspects of the system. In logs, all text printed by the system appears in bold, and all text typed by the user appears in italics. The first few sample runs illustrate the basic functions available to the student user. Later samples illustrate specific types of errors being diagnosed for different phrase models. The last samples indicate the basic functions available to a system administrator user.

3.2. Sample Run

This sample illustrates a normal login, and the correct execution of a single exercise.

```
Sys27> git
German Language Tutor, version 1.0
If you are not sure how to begin, type "?".
Please login: ?
In order to use this system, you must have an authorized username. If you have not received a username yet, see your system administrator. If you do have a valid username, type it in now to gain access to the tutoring system.

Please login: ken
Enter highest chapter completed (currently 5)? 5
Exercise will be limited to chapter 5 and earlier.
You will now be asked to translate various phrases and sentences. You may stop at any time - the important command to remember is "?". Type a question mark at anytime to be told your options and be given the opportunity to leave

Translate: Is the soup warm?
to German: Ist die Suppe warm?
Yes! That is the expected response.

Translate:
```
Also illustrated above is the use of the user knowledge level. The system will not use vocabulary or phrase models from chapters higher than the student indicates.

3.3. Sample Run

This sample illustrates a failed login.
German Language Tutor, version 1.0

If you are not sure how to begin, type "?".

Please login: badname1

That is not a correct username, please verify that you typed what you thought you did, and that you have not used upper or lower case when not expected.

Please login: badname2

That is not a correct username, please verify that you typed what you thought you did, and that you have not used upper or lower case when not expected.

Please login: badname3

Discontinuing attempted login after three tries. Please verify that you are using a correct username, or see the system administrator.

Sys27>

3.4. Sample Run

This run begins in the middle of a session, with an incorrect student response to a translation request. It illustrates the option menu, a student request for help, a student choosing to exit the tutor, and the statistics summary which accompanies program termination.

Translate: here is the glass
to German: hier ist glas

That is not the expected translation. Enter the number indicating the action you would like to take:

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 6

This is a menu of the options that you have at this point. Type the number of the option which you wish to take.

Your choices at this point are to try answering the exercise again, have the system give you a hint about what was wrong, have the system give you the correct answer, go on to a new exercise, exit the system altogether, or view this help message.

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to next.
5 - Exit the tutoring system.
6 - Help.
Choice? 5

German Language Tutor exiting...

Duration of session - 20 minutes
Number of different phrases asked to translate - 50
Number of correct responses - 44
Number of total responses - 57

Sys27>

The summary data indicates that the student was logged into the tutoring system for 20 minutes. During that time, 50 different phrases or sentences were generated for the student to translate. Of that 50, 44 were answered correctly (indicating that the student "quit" or requested the system's answer 6 times). The final count above shows the number of different translation attempts (in this case, 57) that the student entered during the course of the 50 exercises.

3.5. Sample Run

This run begins in the middle of a session, with an incorrect student response to a translation request. It illustrates the user quitting the current exercise, and beginning the next. The next exercise is also answered incorrectly, in such a way that the system cannot perform any intelligent diagnosis. Note that the system does not offer any hints to the user, since it can not make enough of a diagnosis to do so. This run then illustrates the user requesting the system's answer.

Translate: Mr. Jones is an American
to German: Herr Jones ist ein Amerikaner

That is not the expected translation. Enter the number indicating the action you would like to take:

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 5

The system was expecting:

Herr Jones ist Amerikaner

Translate: How old is the child?
to German: this attempt resembles the original in no way

That response is incorrect, but the system is unable to isolate the specific problem.

1 - Try again.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to next.
5 - Exit the tutoring system.
6 - Help.

Choice? 3

The system was expecting:

Wie alt ist das Kind?

Translate:

de Tisch ist rund

That is not the expected translation. Enter the number indicating the action you would like to take:

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

There is something wrong with the first word in your response.

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

The gender of the first word is wrong.

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

The gender of the article must agree with the noun. The noun is masculine, but the gender of the article you used is feminine.

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

The system cannot diagnose this any further.

1 - Try again.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 1

Translate: the table is round
to German: der Mann ist ein Kaufmann

That is not the expected translation. Enter the number indicating the action you would like to take:

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

An article should not be used in this case.

1 - Try again.
2 - Have the system give you another hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to next.
5 - Exit the tutoring system.
6 - Help.

Choice? 1

Translate: the man is a merchant
to German: der Mann ist Kaufmann

3.7. Sample Run

The following sample run illustrates a phrase in which a BasePhraseTranslator was used. In this case, the translation is not literal, it takes a slightly different form. Besides illustrating base phrase translation, this run also shows that the diagnosis tree can be designed to catch the literal translation as a possible student error.

Translate: the man is a merchant
to German: der Mann ist ein Kaufmann

That is not the expected translation. Enter the number indicating the action you would like to take:

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

An article should not be used in this case.

1 - Try again.
2 - Have the system give you another hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to next.
5 - Exit the tutoring system.
6 - Help.

Choice? 1

Translate: the man is a merchant
to German: der Mann ist Kaufmann
3.8. Sample Run

This sample run shows a system administrator login. This user chooses to view diagnosis failures. This would be used as a means for determining how to enhance the system's diagnosis capability. Two different types of diagnosis failures are shown in the dialogue below. The first shows a case where the system successfully isolated the error, and provided hints, but the user asked for additional hints when there were no more to give. The second case shows an answer so different from the expected response, that the system was not even able to make an initial diagnosis. Both of these cases, though, may indicate areas where the system could be improved. (In the second case, this would be true if the actual response was something that should have been isolated by the diagnosis tree.)

German Language Tutor, version 1.0

If you are not sure how to begin, type "?"

Please login: sys_user

Select desired function from menu below.

5 - Exit the tutoring system.
6 - Help.
7 - Write current log entries to file.
8 - Delete log file.
9 - Save log file to new name.
10 - View system diagnosis failures.
11 - Generate report, system diagnosis failures.
12 - View individual past session statistics.
13 - Generate report, individual past session statistics.
14 - View cumulative past session statistics.
15 - Generate report, cumulative past session statistics.
16 - Generate report, all information available.

Choice? 10

Leaf node encountered while looking for additional hints.

Date - Sat Apr 03 1993    Username - stu_user
Index of model chosen - 1
Phrase which was to be translated - the house is old
Expected response - das Haus ist alt
Actual response - der Haus ist alt

5 - Exit the tutoring system.
6 - Help.
4 - Return to main system administrator menu.
17 - View next entry.

Choice? 17

Leaf node encountered while trying to diagnose.

Date - Sat Apr 03 1993    Username - stu_user
Index of model chosen - 2
Phrase which was to be translated - the doctor drinks coffee
Expected response - der Doktor trinkt Kaffee
Actual response - Is this a valid translation attempt?
5 - Exit the tutoring system.
6 - Help.
4 - Return to main system administrator menu.
17 - View next entry.

Choice? 17

No further diagnostic failure entries found.

Select desired function from menu below.

5 - Exit the tutoring system.
6 - Help.
7 - Write current log entries to file.
8 - Delete log file.
9 - Save log file to new name.
10 - View system diagnosis failures.
11 - Generate report, system diagnosis failures.
12 - View individual past session statistics.
13 - Generate report, individual past session statistics.
14 - View cumulative past session statistics.
15 - Generate report, cumulative past session statistics.
16 - Generate report, all information available.

Choice? 5

Your session with the German Language Tutor is complete.

The "index of model chosen" is recorded in the log, in the event that the source code will be entered. This points the programmer to the appropriate phrase model.

3.9. Sample Run

This is another system run, showing a system administrator viewing run-time statistics. The first two sets of statistics are from individual system runs. After returning to the main menu, the system administrator views the cumulative statistics for all statistics entries in the log.

German Language Tutor, version 1.0

If you are not sure how to begin, type "?".

Please login: sys_user

Select desired function from menu below.

5 - Exit the tutoring system.
6 - Help.
7 - Write current log entries to file.
8 - Delete log file.
9 - Save log file to new name.
10 - View system diagnosis failures.
11 - Generate report, system diagnosis failures.
12 - View individual past session statistics.
13 - Generate report, individual past session statistics.
14 - View cumulative past session statistics.
15 - Generate report, cumulative past session statistics.
16 - Generate report, all information available.

Choice? 12

Statistics after a student session.
Date - Sat Apr 03 1993   Username - stu_user
This session lasted 9 minutes, 12 seconds.
Number of different phrases asked to translate - 2
Number of correct translations - 1
Number of total responses - 3

5 - Exit the tutoring system.
6 - Help.
4 - Return to main system administrator menu.
17 - View next entry.

Choice? 17

Statistics after a student session.
Date - Sat Apr 03 1993   Username - stu_user
This session lasted 1 minutes, 14 seconds.
Number of different phrases asked to translate - 3
Number of correct translations - 0
Number of total responses - 3

5 - Exit the tutoring system.
6 - Help.
4 - Return to main system administrator menu.
17 - View next entry.

Choice? 17

No further statistics entries found.

Select desired function from menu below.

5 - Exit the tutoring system.
6 - Help.
7 - Write current log entries to file.
8 - Delete log file.
9 - Save log file to new name.
10 - View system diagnosis failures.
11 - Generate report, system diagnosis failures.
12 - View individual past session statistics.
13 - Generate report, individual past session statistics.
14 - View cumulative past session statistics.
15 - Generate report, cumulative past session statistics.
16 - Generate report, all information available.

Choice? 14

Cumulative statistics from log file.
Number of individual sessions counted - 2
Total duration of sessions - 10 minutes, 26 seconds.
Average duration of sessions - 5 minutes, 13 seconds.
Total number of phrases presented by system - 5
Total number of student responses - 6
Total number of correct translations - 1

5 - Exit the tutoring system.
6 - Help.
4 - Return to main system administrator menu.

Choice? 5

Your session with the German Language Tutor is complete.
4. DESIGN AND IMPLEMENTATION

4.1. Introduction

This chapter summarizes the three major phases in the German Language Tutoring System development: analysis, design, and implementation. There are various object-oriented analysis and design techniques being tried today, many of which borrow elements from one another. This effort relied heavily on two methodologies – Job-Oriented Object Analysis (JOOA) [Nichols, 1991a] and Class-Responsibility-Collaboration (CRC) Design [Wirfs-Brock, 1990]. The methodologies are fairly intuitive, at least once the potential confusion with structured analysis and design concepts are overcome. The documentation should be fairly straightforward, and enough background description has been included to allow the reader unfamiliar with these methodologies to follow the design notes.

In general, object-oriented approaches tend to be more iterative, as opposed to using the more rigid phases and gates of other methodologies. The lines between analysis, design and implementation tend to blur, as earlier decisions are re-visited to address modularity, reusability, and responsibility issues uncovered in later phases. The process does not lend itself to intermediate documentation. Instead, the documentation here represents the more-or-less ”stable state” which was achieved near the end of each phase.

In addition to the development phases mentioned above, the system itself was approached as having two distinct design components. The traditional ”system design” defined the internals of the system – the mechanics of the algorithms, data structures, etc., and the ”knowledge base design” supplied the actual data which drives the system. The system design alone specified an empty shell. The knowledge base design filled in the data. For this type of data-driven processing, the data design is critical. Diagnosis of student responses is done by using trees of patterns to isolate and identify errors. These trees are part of the knowledge base data, and must be carefully chosen to correctly identify errors.

The system design is described beginning immediately below. The knowledge base design is described beginning with section 4.3.

4.2. System Design

4.2.1. System Analysis and Design Goals

The fundamental goal was to produce a working prototype system, experimenting with the use of several concepts. It would be designed, implemented and documented in an object-oriented fashion. In conjunction with this, it was a goal to realize some of the benefits of this approach – an intuitive design, modularity and enhanceability. It was a goal to make the system functional, from a user standpoint, and to include enough
breadth of knowledge to illustrate the various internal features (though not the breadth of knowledge necessary to truly be a marketable product at this point). It was a goal to incorporate the pattern masking technique of diagnosis described earlier in this report, and to support the run-time gathering of data for future improvement of its diagnostic capability.

4.2.2. JOOA: Job-Oriented Object Analysis

JOOA is one of several object-oriented analysis methodologies. It, in turn, attempts to integrate the best concepts from two other methods: "Role Analysis" and "Visualization". It was developed in 1987, and has evolved since then as it has been used for large commercial software development projects. The designer who is familiar with structured, or functional, analysis and design, may find it more difficult to use these methodologies effectively than the designer who is already comfortable with object-oriented programming. Once the basic terminology, notation, and concepts are understood, the approach is fairly intuitive.

The main objective in JOOA is to iteratively identify and refine objects, until they can be described neatly and encompass the specified functionality. It involves approaching the system from two different perspectives of jobs (for which the system is intended) in order that the two provide checks and balances for each other. A brief review of the necessary terminology should be sufficient to allow understanding of the actual analysis report which is included below. Object-oriented documentation, like the design itself, will not necessary convey all necessary information in a single sequential read. However, by referring backward and forward to diagrams and descriptions, the reader should be able to reconstruct the design process.

4.2.2.1. JOOA terminology

There are not too many definitions here, so they have been listed in a more logical, as opposed to alphabetical, order.

Work Place – This is a job view of the system from an external standpoint. In its most literal sense, this would be the actual place where one human works, such as the control panel of a certain machine. Workplaces can also be physically identifiable components of a system, again from an external standpoint (e.g. the feeder, duplicator, and collator components of a copier). In the case of a software system, such as this tutoring system, the workplaces can be the different "hats" that a user of the system might wear (e.g. the teacher, the student, the system administrator or system designer, etc.)

Agent – An agent represents a job step in the system. An easy mistake for the functional designer, is to consider agents to be syn-
onymous with tasks or functions in a structured analysis. In JOOA, an agent is typically at a higher level than this. As the analysis progresses, the agents should become trivial to the point of being just a series of delegations to objects. In fact, many or most agents may go away altogether.

Object – An object is a tangible component of the system. It may represent a physical object (e.g. a "word", in this project's context). An object will have a purpose, some amount of knowledge, and some amount of responsibility (things that it can do to/with its knowledge).

Work Flow – This is a pictorial representation of the flow of objects between agents. Again, the functional designer may confuse these with data flow diagrams. Their true intent is to show agents as black box components, again at a higher level, and show how they are tied together only by the exchange of certain objects. The inputs and outputs of the agents are numbered to represent "pins" in this schematic–like representation of software components. One of the principle reasons for agents and work flows being at a higher and more generic level than tasks and data flows, is to avoid restricting their evolvement. Whereas tasks and data flows tend to be rather stable, and drive the rest of the design, agents and workflows will tend to change as responsibilities are traded between agents and objects as the design coalesces.

Composition Graph – This is the pictorial representation of an object which contains other objects as part of its internal knowledge. The labels on the arrows indicate how many sub–objects are contained, and may be absolute numbers or ranges:

```
  Object1
    ▼1
      0–n
    ▼
     Object2  Object3
```

Begetter Graph – This is a pictorial representation showing objects which create or destroy other objects. It is distinguished from a composition graph by the lack of labels on the arrow. A slash through the arrow indicates object destruction. If no explicit destruction graph is shown, an object is assumed to be destroyed.
by the same object which created it.

Collaboration Graph – This type of graph shows which agents and objects are "customers" of other objects. The numbers within each object box indicate the specific responsibility that the customer is expecting, and are typically described in a key accompanying the graph. This type of graph is usually used in the lower levels of design, but a high-level collaboration graph has been included early in the documentation below to illustrate the Session agent’s context.

There are additional terms used in the JOOA literature, but since this particular design does not require these elements, they have not been included here.

The design of the tutoring system begins with the next section.

4.2.2.2. Work Places and Work Flows

Three workplaces were identified for this system. They are listed below with their basic responsibilities.

Student
   - ID User
   - Quiz User

System Administrator
   - ID User
   - View and Manipulate Logs
Teacher (future) - ID User
- Edit Users
- Edit Knowledge Base

These initial agents were identified from the responsibilities listed above. Since the "Teacher" workplace is a future enhancement, those responsibilities were not included:

![Diagram of object flow](image)

It turns out that this system is not very complex, from this external perspective, so the work flow diagrams are not extremely interesting. The object path shown above represents the flow of a single object, a User.

Normally, the above diagrams would not be included, since they changed in the course of object identification. They have been left above, however, to comment on the process in spite of the fact that only one agent survived intact. This agent, Session, evolved predominantly from the original "QuizUser" agent. Most of the IDUser and GetStatistics responsibilities were delegated to the UserList and Log objects, respectively.

Looking ahead, this is actually an ideal situation, because this single agent could then become the main() function of the C++ implementation. This leaves the final set of Work Flow diagrams for this system as:

![Session diagram](image)

Given the above information, the Work Places are now documented:

**4.2.2.2.1. Student**

This is the primary user of the system. A typical interaction would include logging in, answering some exercises, receiving some feedback, and logging off. A high level of computer competency should not be assumed for this type of user.

**4.2.2.2.1.1. Agents Used**

- Session
4.2.2.2.2. System Administrator

Because this is an experimental system, it will provide for a class of user whose interest will be in system performance, correctness, and other design issues. A higher level of system and computer knowledge may be assumed if necessary.

4.2.2.2.2.1. Agents Used

- Session

4.2.2.3. Agents

This section defines agents which are not part of any workflow (in this case, the sole agent).

4.2.2.3.1. Session

The essential role of this agent is to control the complete login to logout interactive session with a single user.

4.2.2.3.1.1. Responsibilities

- Perform any general system set-up and shut-down.
- Initiate a user login.
- Coordinate series of exercises for the student user.
- Coordinate the access to and manipulation of logged data for an administrative user.

Because there is only a single agent in this design, the following collaboration graph may help illustrate how this agent relates to other major objects in the system.
4.2.2.4. Object Graphs

This section provides an alphabetically listed description of each object uncovered during analysis. Those objects which are necessary for the essential purpose of the system (i.e. user exercises) are described in more detail than other less essential objects (i.e. log manipulation). This is intentional, as it leaves greater design- and implementation-time flexibility in those areas which are less critical.

Before the textual descriptions begin, the composition and begetter graphs are presented. These can be referred back to, in order to understand the relationships between the described objects. The composite illustration below is a visual aid that represents the most complex of the composite objects, the PhraseModel. It conveys roughly the same information as the individual composition graphs (with the exception of the count labels), but it may be helpful to see them together for this complex object. Trivial "composition" graphs for objects which are not currently composed of additional objects have also been included. This adds completion to the illustrations, since all objects are now represented. Additionally, a brief statement of each object's responsibilities has been included next to its graph.
4.2.2.4.1. Composition Graphs

A string of BaseWords representative of a semantically valid phrase. Most likely not syntactically correct.

A collection of BasePhrases of like length which can be made syntactically valid by application of the same PhraseDescriptor.
BasePhrasePool

A collection of BasePhraseLists containing BasePhrases of the same length, and which can be made syntactically valid by application of the same PhraseDescriptor.

BasePhraseList

1–n

BasePhraseTranslator

A set of rules which describes how to translate a specific BasePhrase associated with a specific PhraseModel.

Dictionary

This represents the system’s vocabulary.

0–n

DictionaryEntry

This represents the entire dictionary entry for a single base word (including how to derive other forms of the word, etc.)

DynamicWordDescriptor

Collects attributes of words in a phrase, in order to dynamically set the attributes of later words which must agree with the earlier ones.

Hint

A block of text. Associated with a pattern, it describes the conclusion drawn based on the user’s input matching this pattern.
Log

This object represents all run-time gathered data currently being maintained by the system.

LogEntry

This is represents a single recordable incident. There will be multiple formats that share some common sub-format.

Pattern

This is very similar to a Phrase, but it likely contains wildcards and can match itself against a Phrase.

Word

PatternFragment

Represented the rules for converting a Phrase of a given semantic model to a pattern which will be used for diagnosis (i.e. containing wildcards).

PatternConstructor

PatternFragmentConstructor

This is a pattern primitive.
PatternFragmentConstructor

Describes how to modify a Word in a phrase. This is a pattern-building tool, where patterns containing wildcards are constructed from existing Phrases.

PatternNode

This is a single entry in a binary tree of Patterns which is used to process user responses. It contains information on how to construct the current pattern, as well as which Patterns to move on to, based on whether or not the user response matches at this point.

Phrase

This is an actual semantically and syntactically correct text phrase which can be printed to the screen.

PhraseDescriptor

A string of WordDescriptors representative of a syntactically valid phrase. A Base-Phrase and a PhraseDescriptor contain all the information necessary to create a syntactically and semantically correct text phrase.
This models everything that the system knows about a specific semantic model phrase – what base phrases may be used in it, how to translate it, how to parse and diagnose the user's response, etc.

A collection of all the model phrases which the system currently supports.

This object will collect and process various run-time statistics (number of exercises completed, etc.)
TranslationTree

This object contains everything necessary to diagnose responses for a particular PhraseModel. It knows how to build the translation phrase, and holds the root pattern of the diagnosis tree.

User

This may eventually be a composite object, but at this level of the design it works fine as a base object. This represents everything that needs to be known about a given user - the login name, the user type (student, system designer, teacher), etc.

UserList

This object represents all the valid users who can log onto the system.

Word

This is an actual text word which can be printed to the screen or used to build phrases.

WordDescriptor

A collection of attributes which specify what derivative of a word is needed. (Not the rules, just the attributes, e.g. 'plural', 'present tense', etc.)
4.2.2.4.2. Begetter Graphs

A PhraseDescriptor can create a Phrase when given a BasePhrase to work from.

A BasePhraseTranslator creates a BasePhrase in the opposite language.

A PatternConstructor can create a Pattern when given a Phrase to work from.

Given desired featured information, the DictionaryEntry will create the appropriate derived form of the desired word.

The Log creates individual entry objects.
A Session creates objects to represent the current User, current session Statistics, the UserInterface, and the run–time data Log.

The TranslationTree destroys the translated BasePhrase when finished.

The PhraseModel destroys the presentation phrase once the exercise is completed.

The TranslationTree destroys the user response phrase once it is finished.

The PatternNode destroys the dynamically created pattern once it is done trying to match.
4.2.2.5. Object Descriptions

Each object is described individually below. They appear in alphabetical order.

4.2.2.5.1. BasePhrase

The essential role of this object is to specify a semantically valid phrase. Most likely not syntactically correct.

4.2.2.5.1.1. Responsibilities

- specify a phrase, but not the attributes of the words used.

4.2.2.5.1.2. Attributes

- words – The Dictionary identifiers for the base words in this phrase.
- length – The number of words in this phrase.
- active – a flag which indicates whether or not this phrase is within the current user’s knowledge level.

4.2.2.5.1.3. Actions

- get_word_identifier(position) – Used when building syntactically correct phrases to obtain words.
- set_active(know_level) – Sets the active flag if the vocabulary in this phrase is within the user’s knowledge level.

4.2.2.5.2. BasePhraseList

The essential role of this object is to collect BasePhrases of like length which can be made syntactically valid by application of the same Phrase Descriptor.

4.2.2.5.2.1. Responsibilities

- collect, and allow selection of a random BasePhrase.

4.2.2.5.2.2. Attributes

- phrases – The collection of BasePhrases which make up this list.
- size – The number of phrases in this list.
- selected – An indication of which BasePhrases have been used in the current session.
- active – a flag which indicates whether or not any of the phrases in this list are within the user’s knowledge level.

4.2.2.5.2.3. Actions

- get_base_phrase() – gets random base phrase from list.
• reset_selections() – Clean the slate of phrases marked as ‘used’. Done to start over after all have been used.

• set_active(know_level) – Sets the active flag if any of the BasePhrases in this list are within the user’s knowledge level.

4.2.2.5.3. BasePhrasePool

The essential role of this object is to collect all BasePhraseLists containing BasePhrases of the same length, and which can be made syntactically valid by application of the same PhraseDescriptor. Note that this is very similar to the BasePhraseList. The reason that they both exist is that different subsets of the BasePhrasePool (i.e. BasePhraseLists) may be applicable to different PhraseModels, and are then gathered into BasePhrasePools.

4.2.2.5.3.1. Responsibilities

• collect, and allow selection of a random BasePhrase.

4.2.2.5.3.2. Attributes

• phrase_lists – The collection of BasePhraseLists which make up this pool.

• size – The number of BasePhraseLists in the pool.

• depleted – An indication of which BasePhraseLists have been exhausted in the current session.

• active – a flag which indicates whether or not any of the lists in this pool contain phrases which are within the user’s knowledge level.

4.2.2.5.3.3. Actions

• get_base_phrase() – returns random base phrase from pool.

• reset_selections() – Clean the slate of BasePhraseLists marked as ”depleted”. Done to start over after all have been used.

• set_active(know_level) – Sets the knowledge level of the system to match that of the current user.

4.2.2.5.4. BasePhraseTranslator

The essential role of this object is to specify the rules which describe how to translate a specific BasePhrase associated with a specific PhraseModel.

4.2.2.5.4.1. Responsibilities

• translate a BasePhrase, into a BasePhrase of the other language.

4.2.2.5.4.2. Attributes

• rules – a set of translation actions, specifically designed to translate a BasePhrase from one language to the other. (e.g. this may involve adding or deleting words, changing the word order, etc.)
4.2.2.5.4.3. Actions

- `translate(BasePhrase)` – returns the translation.

### 4.2.2.5.5. Dictionary

The essential role of this object is to represent the system’s vocabulary – all the words that the system knows.

#### 4.2.2.5.5.1. Responsibilities

- service requests for forms of BaseWords.

#### 4.2.2.5.5.2. Attributes

- entries – the collection of individual DictionaryEntries which make up the Dictionary.

#### 4.2.2.5.5.3. Actions

- `get_word(word_identifier, WordDescriptor)` – returns the derived form of the indicated word, as specified by the attributes in the WordDescriptor.

### 4.2.2.5.6. DictionaryEntry

The essential role of this object is to represent a given word in the dictionary. It represents all that the dictionary knows about a single base word (including how to construct other forms of the word, etc.)

#### 4.2.2.5.6.1. Responsibilities

- service requests for information or derived forms of BaseWords.

#### 4.2.2.5.6.2. Attributes

- `word_identifier` – A unique key used to identify words in the dictionary, and represent them elsewhere in the system.
- `word_attributes` – The attributes of a word which are pre-defined, such as the gender of a given noun.
- `derivations` – The derived variations of the base word.

#### 4.2.2.5.6.3. Actions

- `derive_word(WordDescriptor)` – Applies necessary morphological rules to return the derived form of the word as specified by the attributes in the WordDescriptor.
- `get_identifier()` – Return the word’s key.

### 4.2.2.5.7. DynamicWordDescriptor

The essential role of this object is to facilitate the propagation of word attributes for words which must agree with one another.
4.2.2.5.7.1. Responsibilities
- Accumulate attributes and resolve dynamic attributes for individual Words.

4.2.2.5.7.2. Attributes
- word_attributes – The list of attributes (gender, etc.) which may need to agree.

4.2.2.5.7.3. Actions
- get_attributes() – Return the current set of attributes.
- resolve_dynamics(Word) – Get any needed attributes from the input Word.

4.2.2.5.8. Hint
The essential role of this object is to assist the student when asked, or in certain error cases. It is basically a block of text, associated with a given Pattern.

4.2.2.5.8.1. Responsibilities
- Draw some conclusions for the user based on current response.

4.2.2.5.8.2. Attributes
- text.

4.2.2.5.8.3. Actions
- print() – display self to screen.

4.2.2.5.9. Log
The essential role of this object is to store and service accesses to run–time logged data.

4.2.2.5.9.1. Responsibilities
- store data.
- add (log) data.
- manipulate data.

4.2.2.5.9.2. Attributes
- count – the number of log entries being stored.
- entries – the list of log entries.
- current_data – some data will be maintained at this level for the current session (e.g. current_user).

4.2.2.5.9.3. Actions
- add_entry(entry data) – log data.
• set_current_data(data) – the log will keep track of some data which does not change on an entry-to-entry basis, e.g. current_user)
• manipulate_entries(commands) – various run-time viewing and report-generating functions.

4.2.2.5.10. LogEntry
The essential role of this object is to capture a significant piece of data for further review. There will be various forms of log entries, with data specific to the type of incident being logged.

4.2.2.5.10.1. Responsibilities
• Save, report, or otherwise process a single recordable piece of system information.

4.2.2.5.10.2. Attributes
• record_type – Identifies the incident-specific data format below.
• username – The current user at the time the data was logged.
• date_time – Date/time stamp, when the incident occurred.
• model – The PhraseModel which was being used at the time.
• type-specific_data – The information which will be significant to the system administrator/designer who will be reviewing it.

4.2.2.5.10.3. Actions
• print() – Prints data in some pre-established format.
• report(type) – Report-generation functions.

4.2.2.5.11. Pattern
This is a Phrase that contains wildcards.

4.2.2.5.11.1. Responsibilities
• Can match itself against a Phrase.

4.2.2.5.11.2. Attributes
• fragments – the string of words and/or wildcards which comprise the phrase.
• size – the number of fragments in this pattern.

4.2.2.5.11.3. Actions
• match(Phrase) – Return success or fail depending on whether or not the input Phrase is matched by this pattern.
• get_word(position) – Return the Word at the specified position.
• set_word(position, Word) – Defines the Word at the specified position.

4.2.2.5.12. PatternConstructor

The essential role of this object is to generate a pattern from a given phrase. It may contain PatternFragmentConstructors for more detailed pieces using Words.

4.2.2.5.12.1. Responsibilities
• build Pattern from Phrase.

4.2.2.5.12.2. Attributes
• rules – the steps needed to convert the expected Phrase form to a Pattern.

4.2.2.5.12.3. Actions
• build_pattern(Phrase) – modify the given text phrase to get a pattern with wild-cards.

4.2.2.5.13. PatternFragmentConstructor

The essential role of this object is to generate a pattern fragment from a given Word. It contains rules specifying how much of the word to use, what wild cards to inject, etc.

4.2.2.5.13.1. Responsibilities
• create pattern fragment from Word.

4.2.2.5.13.2. Attributes
• rules – a set of actions specifically designed to change a given Word. It may replace the entire Word with a wildcard, replace certain letter(s), delete the Word altogether, etc.

4.2.2.5.13.3. Actions
• build_pattern_fragment(Word) – modify the given Word.

4.2.2.5.14. PatternNode

The essential role of this object is to recognize and handle a specific type of error in a student’s response. It will be a single entry in a binary tree of Patterns which support a given PhraseModel. It contains information on how to construct the needed matching Pattern, as well as which PatternNodes to move on to, based on whether or not the user response matches at this point.

4.2.2.5.14.1. Responsibilities
• diagnose and act on a student’s response.
4.2.2.5.14.2. Attributes

- **pattern_descriptor** – A PatternConstructor which describes how to construct the actual current Pattern (complete with wildcards, etc.) from the translation phrase.
- **hint** – The hint which can be given if the user input matches this pattern (note, the user’s input is already incorrect, if the system has reached the point of matching against patterns.)
- **match_pattern** – The next PatternNode to try if the user input matches the current pattern and the user wants further diagnosis.
- **no-match_pattern** – The next PatternNode to try if the user input does not match the current pattern (the system must keep trying.)

4.2.2.5.14.3. Actions

- **process_response(expected_response, actual_response)** – builds the necessary pattern, attempts to match the user input, and continue to subsequent no-match PatternNodes if necessary.
- **more_hints(expected_response, actual_response)** – very similar to process_response(), but follows the match PatternNode path at the user’s request for additional hints.

4.2.2.5.15. Phrase

This is an actual semantically and syntactically correct text phrase which can be printed to the screen. Note that a BasePhrase and PhraseDescriptor fully specify a given Phrase.

4.2.2.5.15.1. Responsibilities

- a collection of words.

4.2.2.5.15.2. Attributes

- **size** – the number of words in this phrase.
- **words** – the string of words which comprise the phrase.

4.2.2.5.15.3. Actions

- **print()** – Print self to screen.
- **read()** – Read a phrase from the user.
- **get_word(position)** – Return the Word at the specified position.
- **set_word(position, Word)** – Defines the Word at the specified position.

4.2.2.5.16. PhraseDescriptor

The essential role of this object is to specify the syntactic requirements of a given Phrase. A BasePhrase and a PhraseDescriptor contain all the information necessary to create a semantically and syntactically correct text Phrase.
4.2.2.5.16.1. Responsibilities
• create a Phrase from a BasePhrase.

4.2.2.5.16.2. Attributes
• word_descriptor_list – the WordDescriptor for constructing each word in the desired phrase.
• size – the number of WordDescriptors in this PhraseDescriptor.
• fill-in_order – The order in which to generate the resulting Phrase, so that words that must agree with each other do.

4.2.2.5.16.3. Actions
• build_phrase(BasePhrase) – Constructs the syntactically correct phrase from the semantically correct BasePhrase.

4.2.2.5.17. PhraseModel
The essential role of this object is to model everything that the system knows about a specific semantic model phrase – what BasePhrases may be used in it, how to translate it, how to parse and diagnose the user’s response, etc.

4.2.2.5.17.1. Responsibilities
• construct and process an exercise based on this model.

4.2.2.5.17.2. Attributes
• active – a flag which indicates whether or not this model is within the current user’s knowledge level.
• pool – The BasePhrasePool which this model can draw from.
• fixed_phrase_descriptor – The PhraseDescriptor which contains the constant attributes.
• translation_tree – the root of a pattern tree used to diagnose user inputs for this model.

4.2.2.5.17.3. Actions
• set_active(know_level) – Sets the knowledge level of the system to match that of the current user.
• do_exercise() – Chooses random BasePhrase, constructs semantically correct phrase for user, reads user response, gives to pattern tree for diagnosis.
• reset_depletions() – Clear any internal data which has been marked "used" or "depleted" (e.g. BasePhrases, etc.)

4.2.2.5.18. PhraseModelList
The essential role of this object is to collect all the semantic model phrases (plus associated exercises, etc.) which the system currently supports.
4.2.2.5.18.1. Responsibilities
   - coordinate exercises from different PhraseModels.

4.2.2.5.18.2. Attributes
   - models – the collection of PhraseModels which make up the list.
   - size – number of PhraseModels in the list.
   - depleted – indicates which PhraseModels have been "depleted" (i.e. all vocabulary selections used in the current session).

4.2.2.5.18.3. Actions
   - set_active(know_level) – Sets the knowledge level of the system to match that of the current user.
   - do_exercise() – Chooses a random model for a single translation exercise to be asked of the user.
   - reset_depletions() – Mark all data as "OK to use" again.

4.2.2.5.19. Statistics
   The essential role of this object is to do run-time statistics gathering and processing.

4.2.2.5.19.1. Responsibilities
   - gather, summarize, and print statistics.

4.2.2.5.19.2. Attributes
   - various run-time data counts (e.g. number of exercises attempted)

4.2.2.5.19.3. Actions
   - update_counts(new counts) – Increment run-time data counts
   - print_summary() – Make any statistical conclusions and print.

4.2.2.5.20. TranslationTree
   This object is the beginning of the entire diagnosis-side of a given PhraseModel.

4.2.2.5.20.1. Responsibilities
   - Generates expected response.
   - Reads user responses.
   - Begins diagnosis.

4.2.2.5.20.2. Attributes
   - pattern_root – This is the PatternNode which represents the root of the binary diagnosis tree.
• base_phrase_trans – This is the BasePhraseTranslator which generates the semantically (but probably not syntactically) correct expected response.
• phrase_descrip – This is the syntactic information for the expected response.

4.2.2.5.20.3. Actions
• check_response(base_phrase) – Performs all of the above responsibilities.

4.2.2.5.21. User

The essential role of this object is to represent and maintain all that is known about an individual authorized to use the system.

4.2.2.5.21.1. Responsibilities
• Supply user information.
• Confirm user information.

4.2.2.5.21.2. Attributes
• name – The user’s full name.
• know_level – The student’s former (or current, if logged in) knowledge level.
• user_id – The login name of this user.
• user_type – Student, system designer or teacher (future).

4.2.2.5.21.3. Actions
• get_type() – Returns user_type.
• get_know_level() – Returns know_level.
• set_know_level() – Changes know_level.
• match_user(user_id) – See if the input id matches this user’s.
• get_user() – Returns the user_id of this user.

4.2.2.5.22. UserInterface

The essential role of this object is to centralize all I/O in order to maintain a consistent external interface, and to facilitate future upgrade to a more elaborate interface.

4.2.2.5.22.1. Responsibilities
• Output to screen and input from keyboard.

4.2.2.5.22.2. Attributes
• none.

4.2.2.5.22.3. Actions
• get_user_response() – Read a user’s translation attempt.
• print_menu() – Offer a menu of choices to the user.
• get_menu_response() – Read a user’s menu choice.

4.2.2.5.23. UserList
The essential role of this object is to represent and maintain all the valid users who can log onto the system.

4.2.2.5.23.1. Responsibilities
• coordinate/validate an attempted login.
• add/delete users.
• [supply user information?].

4.2.2.5.23.2. Attributes
• users – the collection of User objects which make up the list.
• size – the number of users currently authorized.

4.2.2.5.23.3. Actions
• establish connection() – Accepts an attempted login, validates that login name is in list, if student, queries knowledge level.
• add_user() – Creates a new User object, and adds it to the collection.
• delete_user(User) – Deletes a User object from the collection.

4.2.2.5.24. Word
This is an actual derived word which can be printed to the screen. Note that a BaseWord and WordDescriptor fully specify a given Word.

4.2.2.5.24.1. Responsibilities
• Maintain the text of the word and pertinent attributes.

4.2.2.5.24.2. Attributes
• string – the string of characters which comprise the word.
• word_attributes – saved along with the string for when another object needs to know.
• word_identifier – the dictionary key for this word.

4.2.2.5.24.3. Actions
• print() – Print self to screen.

4.2.2.5.25. WordDescriptor
The essential role of this object is to specify the attributes which indicate what derivative of a BaseWord is needed. (Not the rules, just the attributes, e.g. "plural", "present tense", etc.)
4.2.2.5.25.1. Responsibilities

- create derived Word from BaseWord.

4.2.2.5.25.2. Attributes

- word_attributes – these are specific to different types of words, for instance, a noun would have "gender", a verb would have "tense", etc.

4.2.2.5.25.3. Actions

- no_dynamics() – returns TRUE or FALSE depending on whether or not all of the attributes in the descriptor are specified.
- resolve_dynamics(DynamicWordDescriptor) – Attempts to define any unspecified attributes by using those in the input descriptor.
- get_attributes() – return the attribute settings.
- set_attributes(attributes) – set the explicitly specified input attributes.

4.2.3. CRC: Class-Responsibility-Collaboration Design

The CRC method is sometimes referred to as the 3W method, in reference to the authors Wirfs-Brock, Wilkerson and Wiener [Wirfs-Brock, 1990]. This method can be summarized as the following:

- Identify the Classes in the system. This consists of further refinement and definition of the objects identified during the JOOA process.
- Determine the Responsibilities of each class. This is the continuing, iterative arbitration of responsibilities between the different classes.
- Determine which, and how, objects of these classes Collaborate with others. This is done in conjunction with the arbitration process, determining how classes interact, and what capabilities are provided to others, as the responsibilities stabilize. Classes may be grouped into functional subsystems, as well.

4.2.3.1. CRC terminology

Refer also back to section 4.2.2.1. for JOOA terminology, if needed.


Class – This is a template or description of a type of object. A class describes all of the data and actions that an object of that class would have. Instances of a class are the actual objects.


Contract – The set of responsibilities which one class (server) provides for another class (client). The contract typically repre-
sents the set of member functions used by a client class, and it should be noted that a given class can be both a client and a server (to the same or different classes).

Inheritance – A class is said to "inherit" from another class when its definition is based on the first class, and as a result, it picks up some of the same attributes or functions. The class which inherits is referred to as a Derived Class, and the class it inherits from is referred to as a Base Class.

Inheritance Graph – This is the pictorial representation of classes which inherit attributes or functionality from other classes. A darkened corner on a class box indicates that it strictly exists to be inherited from, and that no actual objects of that class will ever be created (this is a virtual base class).

Responsibility – In this context, this refers to the responsibilities delegated to particular classes.

Collaboration – The way in which different classes work together to get the overall job done.

Subsystem – A logical collection of classes which, from some client perspective, can be considered as being a single object.

4.2.3.2. Subsystems (in alphabetical order)

4.2.3.2.1. Dictionary Subsystem

4.2.3.2.1.1. Purpose

• Supplies words to the rest of the system.

4.2.3.2.1.2. Classes Used

• Dictionary, The derived classes of DictionaryEntry – DE_Noun, DE_Verb, DE_Adjective, DE_Article.

Responsibility
Get a specified word

Delegated to
Dictionary
Get the derived form of the specified word

DictionaryEntry
(and derived classes)

4.2.3.2.2. Exercises Subsystem

4.2.3.2.2.1. Purpose

• Provides exercise capability.

4.2.3.2.2.2. Classes Used

• PhraseModelList, PhraseModel, BasePhrasePool, BasePhraseList, BasePhrase, BasePhraseTranslator, TranslationTree, PatternNode, Hint, PhraseDescriptor.

Responsibility
External interface, collects and organizes all exercise support
Delegated to
PhraseModelList

A single phrase or sentence form, exercises and diagnosis
PhraseModel

Supplying a random BasePhrase from multiple lists
BasePhrasePool

Supplying a random BasePhrase
BasePhraseList

Semantically correct vocabulary information
BasePhrase

Supplying a BasePhrase of the opposite language
BasePhraseTranslator

Construction of expected response and initial diagnosis
TranslationTree

Subsequent diagnosis
PatternNode

A clue to the discrepancy at a given level
Hint

Syntactically correct model information
PhraseDescriptor

4.2.3.2.3. Log Subsystem

4.2.3.2.3.1. Purpose

• Storing, viewing and manipulating run–time gathered data.

4.2.3.2.3.2. Classes Used

• Log, The derived classes of LogEntry – LE_LeafNode, LE_Statistics.

Responsibility
Record data
Delegated to
Log

View data
Log

Store and manipulate a specific data record
LogEntry
(and derived classes)
4.2.3.3. Classes (in alphabetical order)

4.2.3.3.1. BasePhrase

4.2.3.3.1.1. Purpose

• Specifies a semantically valid phrase. Most likely not syntactically correct.

4.2.3.3.1.2. Base Classes

• None.

4.2.3.3.1.3. Derived Classes

• None.

Responsibility  Collaborators
get_word_identifier  none
set_active  none

4.2.3.3.2. BasePhraseList

4.2.3.3.2.1. Purpose

• Collects BasePhrases of like length which can be made syntactically valid by application of the same Phrase Descriptor

4.2.3.3.2.2. Base Classes

• None.

4.2.3.3.2.3. Derived Classes

• None.

Responsibility  Collaborators
get_base_phrase  none
reset_selections  none
set_active  none

4.2.3.3.3. BasePhrasePool

4.2.3.3.3.1. Purpose

• Collects all BasePhraseLists containing BasePhrases of the same length, and which can be made syntactically valid by application of the same Phrase Descriptor.

4.2.3.3.3.2. Base Classes

• None.
4.2.3.3.3. Derived Classes

- None.

**Responsibility**

get_base_phrase
reset_selections
set_active

**Collaborators**
BasePhraseList

4.2.3.3.4. BasePhraseTranslator

4.2.3.3.4.1. Purpose

- Specifies the rules which describe how to translate a specific BasePhrase associated with a specific PhraseModel.

4.2.3.3.4.2. Base Classes

- None.

4.2.3.3.4.3. Derived Classes

- None.

**Responsibility**

translate

**Collaborators**
BasePhrase
Dictionary

4.2.3.3.5. Dictionary

4.2.3.3.5.1. Purpose

- Represents the system’s vocabulary – all the words that the system knows.

4.2.3.3.5.2. Base Classes

- None.

4.2.3.3.5.3. Derived Classes

- None.

**Responsibility**

get_word

**Collaborators**
DictionaryEntry

4.2.3.3.6. DictionaryEntry

4.2.3.3.6.1. Purpose

- Represents a given word in the dictionary. It represents all that the dictionary knows about a single base word (including how to construct other forms of the word, etc.)
4.2.3.3.6.2. Base Classes

- None.

4.2.3.3.6.3. Derived Classes

- DE_Noun, DE_Verb, DE_Article, DE_Adjective.

Responsibility | Collaborators
--- | ---
derive_word | Word
get_identifier() – Return the word’s key. | none

4.2.3.3.6.4. Notes

- Additional derived forms will be added and documented as the system is enhanced.

4.2.3.3.7. DynamicWordDescriptor

4.2.3.3.7.1. Purpose

- Facilitates the propagation of word attributes for words which must agree with one another.

4.2.3.3.7.2. Base Classes

- None.

4.2.3.3.7.3. Derived Classes

- None.

Responsibility | Collaborators
--- | ---
get_attributes | none
resolve_dynamics | Word

4.2.3.3.8. Hint

4.2.3.3.8.1. Purpose

- Assists the student when asked, or in certain error cases. It is basically a block of text, associated with a given Pattern.

4.2.3.3.8.2. Base Classes

- None.

4.2.3.3.8.3. Derived Classes

- None.
4.2.3.3.9. Log

4.2.3.3.9.1. Purpose
   • Stores and services accesses to run-time logged data.

4.2.3.3.9.2. Base Classes
   • None.

4.2.3.3.9.3. Derived Classes
   • None.

4.2.3.3.10. LogEntry

4.2.3.3.10.1. Purpose
   • Captures a significant piece of data for further review. There will be various forms of log entries, with data specific to the type of incident being logged.

4.2.3.3.10.2. Base Classes
   • None.

4.2.3.3.10.3. Derived Classes
   • LE_Statistics, LE_LeafNode

Responsibility

print

Collaborators

UserInterface

add_entry

LogEntry

set_current_user

none

set_current.presentation_phrase

Phrase

set_current_model_index

none

manipulate_entries

LogEntry

save

LogEntry

4.2.3.3.10

report

Statistics

Phrase

Statistics

Phrase
4.2.3.3.10.4. Notes

- Additional derived forms will be added and documented as the system is enhanced.

4.2.3.3.11. Pattern

4.2.3.3.11.1. Purpose

- This is essentially a Phrase that contains wildcards, and which can match itself against other phrases.

4.2.3.3.11.2. Base Classes

- None.

4.2.3.3.11.3. Derived Classes

- None.

Responsibility

match
get_word
set_word

Collaborators

4.2.3.3.12. PatternConstructor

4.2.3.3.12.1. Purpose

- Generates a pattern from a given phrase. It may contain PatternFragmentConstructors for more detailed pieces using Words.

4.2.3.3.12.2. Base Classes

- None.

4.2.3.3.12.3. Derived Classes

- None.

Responsibility

build_pattern

Collaborators

PatternFragmentConstructor
Dictionary
Phrase
Word

4.2.3.3.13. PatternFragmentConstructor

4.2.3.3.13.1. Purpose

- Generates a pattern fragment from a given Word. It contains rules specifying how much of the word to use, what wild cards to inject, etc.
4.2.3.3.13.2. Base Classes

- None.

4.2.3.3.13.3. Derived Classes

- None.

Responsibility

\texttt{build\_pattern\_fragment}

4.2.3.3.14. PatternNode

4.2.3.3.14.1. Purpose

- Recognizes and handles a specific type of error in a student’s response. It will be a single entry in a binary tree of Patterns which support a given PhraseModel. It contains information on how to construct the needed matching Pattern, as well as which PatternNodes to move on to, based on whether or not the user response matches at this point.

4.2.3.3.14.2. Base Classes

- None.

4.2.3.3.14.3. Derived Classes

- None.

Responsibility

\texttt{process\_response}

4.2.3.3.15. Phrase

4.2.3.3.15.1. Purpose

- This is an actual semantically and syntactically correct text phrase which can be printed to the screen. Note that a BasePhrase and PhraseDescriptor fully specify a given Phrase.
4.2.3.3.15.2. Base Classes

- None.

4.2.3.3.15.3. Derived Classes

- None.

**Responsibility**

print
read
get_word
set_word

4.2.3.3.16. PhraseDescriptor

4.2.3.3.16.1. Purpose

- Specifies the syntactic requirements of a given Phrase. A BasePhrase and a PhraseDescriptor contain all the information necessary to create a semantically and syntactically correct text Phrase.

4.2.3.3.16.2. Base Classes

- None.

4.2.3.3.16.3. Derived Classes

- None.

**Responsibility**

build_phrase

4.2.3.3.17. PhraseModel

4.2.3.3.17.1. Purpose

- Models everything that the system knows about a specific semantic model phrase – what BasePhrases may be used in it, how to translate it, how to parse and diagnose the user’s response, etc.

4.2.3.3.17.2. Base Classes

- None.

4.2.3.3.17.3. Derived Classes

- None.
Responsibility

set_active

do_exercise

reset_depletions

Collaborators

BasePhrasePool

PhraseDescriptor

Log

Statistics

Phrase

TranslationTree

BasePhrasePool

4.2.3.3.18. PhraseModelList

4.2.3.3.18.1. Purpose

- Collects all the semantic model phrases (plus associated exercises, etc.) which the system currently supports.

4.2.3.3.18.2. Base Classes

- None.

4.2.3.3.18.3. Derived Classes

- None.

Responsibility
do_exercise

set_active

reset_depletions

Collaborators

PhraseModel

PhraseModel

PhraseModel

4.2.3.3.19. Statistics

4.2.3.3.19.1. Purpose

- Does run–time statistics gathering and processing.

4.2.3.3.19.2. Base Classes

- None.

4.2.3.3.19.3. Derived Classes

- None.

Responsibility

update_counts

Collaborators

none
4.2.3.3.20. TranslationTree

4.2.3.3.20.1. Purpose
   • This object is the beginning of the entire diagnosis-side of a given PhraseModel.

4.2.3.3.20.2. Base Classes
   • None.

4.2.3.3.20.3. Derived Classes
   • None.

  Responsibility   Collaborators
  check_response  BasePhraseTranslator
                  PhraseDescriptor
                  Phrase
                  Statistics
                  Pattern
                  PatternNode

4.2.3.3.21. User

4.2.3.3.21.1. Purpose
   • Represents and maintains all that is known about an individual authorized to use
     the system.

4.2.3.3.21.2. Base Classes
   • None.

4.2.3.3.21.3. Derived Classes
   • None.

  Responsibility   Collaborators
  get_type        none
  get_know_level  none
  set_know_level  none
  match_user      none
  get_user        none

4.2.3.3.22. UserInterface

4.2.3.3.22.1. Purpose
   • Centralizes all I/O in order to maintain a consistent external interface, and to fa-
     cilitate future upgrade to a more elaborate interface.
4.2.3.3.22.2. Base Classes

- None.

4.2.3.3.22.3. Derived Classes

- None.

Responsibility Collaborators
get_user_response Phrase
print_menu none
get_menu_response none

4.2.3.3.23. UserList

4.2.3.3.23.1. Purpose

- Represents all the valid users who can log onto the system.

4.2.3.3.23.2. Base Classes

- None.

4.2.3.3.23.3. Derived Classes

- None.

Responsibility Collaborators
establish connection User
add_user User
delete_user User

4.2.3.3.24. Word

4.2.3.3.24.1. Purpose

- This is an actual derived word which can be printed to the screen. Note that a BaseWord and WordDescriptor fully specify a given Word.

4.2.3.3.24.2. Base Classes

- None.

4.2.3.3.24.3. Derived Classes

- None.

Responsibility Collaborators
print none
4.2.3.25. **WordDescriptor**

4.2.3.25.1. **Purpose**
- Specifies the attributes which indicate what derivative of a `BaseWord` is needed.
  (Not the rules, just the attributes, e.g. "plural", "present tense", etc.)

4.2.3.25.2. **Base Classes**
- None.

4.2.3.25.3. **Derived Classes**
- None.

**Responsibility**
- `no_dynamics`
- `resolve_dynamics`

**Collaborators**
- `DictionarySubsystem`
- `DynamicWordDescriptor`

4.2.3.4. **Collaboration Graphs**

4.2.3.4.1. **DictionarySubsystem**
4.2.3.4.2. Exercises Subsystem

Exercises Subsystem

12. PhraseDescriptor

12.1

3. PhraseModelList

4. PhraseModel

5. TranslationTree

6. PatternNode

7. Hint

11. BasePhraseTranslator

11.1

1. PhraseModelList

2

3.1

4.1

5.1

6.1

7.1

8. BasePhrasePool

8.1

9. BasePhraseList

9.1

10. BasePhrase

10.1
4.2.3.4.3. Log Subsystem

4.2.3.5. Contract Listings

4.2.3.5.1. Subsystem Contracts
(Numbers refer to Collaboration Graphs above.)

1. Get word from dictionary.
2. Set active knowledge level, do exercise.
3. Set current session information, log data, view data, generate report.

4.2.3.5.2. Class Contracts in Subsystems
(Numbers refer to Collaboration Graphs above.)

1. Dictionary
   1.1 Get word from dictionary.
2. DictionaryEntry
   2.1 Get index, derive word.
3. PhraseModelList
   3.1 Set active knowledge level, do exercise.
4. PhraseModel
   4.1 Set active knowledge level, do exercise, reset depletions
5. TranslationTree
   5.1 Check student response.
6. PatternNode
6.1 Process response.
7. Hint
   7.1 Print.
8. BasePhrasePool
   8.1 Set active knowledge level, get BasePhrase, reset selections.
9. BasePhraseList
   9.1 Set active knowledge level, get BasePhrase, reset selections.
10. BasePhrase
   10.1 Get index.
11. BasePhraseTranslator
   11.1 Translate.
12. PhraseDescriptor
   12.1 Build Phrase.
13. Log
   13.1 Set current session information, log data, view data, generate report.
14. LogEntry
   14.1 Print.

4.2.3.6. Inheritance Graphs

4.2.4. Degree of Analysis and Design Goal Satisfaction

The design was completed and documented in an object-oriented fashion. What is documented here is the result of many iterations. The design supports all of the specified
functionality, and the C++ implementation was fairly straightforward. It is not clear that all of the touted benefits of object-oriented design have been realized. It is quite modular, and as intuitive as the chosen parsing algorithms allow it to be. It would seem to lend itself to enhancement. Reusability (or lack thereof) is one area where the design may fall short. Along the way, decisions were made on what responsibilities objects would need to support, and what the division of objects would be. It was not until the implementation phase, though, that the pervasiveness of some of the concepts of the system were appreciated. The knowledge level, for instance, touches nearly every object in one way or another. It would have been nice to isolate such tutoring system-specific needs from the generic word, phrase, etc., processing. Tim Nichols, who promotes the JOOA methodology, advises beginning object designers to tackle a project strictly as a learning exercise, and then, once it reaches the implementation phase, scrap it entirely and chalk up the time to the design learning curve. It would be nice to have that luxury, because a new design would surely come out much cleaner.

4.3. Knowledge Base Design

It was a design goal for this system to support the gathering of data which would allow its diagnosis ability to be improved. Since this is the best way to find out what type of responses will be encountered, it is not unlikely that, over time, the internal knowledge base reflect more of this feedback than of the original design. This is not unreasonable, or unexpected, but it is still necessary to put in place an initial knowledge base from which to work. This involved identifying several things: the phrase/sentence models to be supported, the vocabulary to be supported for different levels, the types of errors which the system would be able to diagnose for a given model, and the processing details to support the different types of errors. The following sections describe the process of establishing the initial knowledge base and knowledge-specific processing.

4.3.1. Knowledge Base Design Goals

As alluded to above, the primary goal of the initial knowledge design was to put something in place which would allow the system to be demonstrated, run, and capable of gathering data for future enhancement. It was also a goal to have the diagnosis be correct (for those errors handled), and for multiple levels of user knowledge to be supported.

4.3.2. Phrase Model Selection

As stated earlier, this incarnation of the tutoring system is highly tied to a specific textbook. It was also decided that "chapter completed" would be a reasonable definition of knowledge level, at least for the purpose of implementing the prototype system. The textbook chosen was German Made Simple [Jackson, 1985]. This choice was made for several reasons. It was the textbook for two introductory German courses taken as information gathering for this project. This made it a common denominator for the sys-
tem designer, the course instructor, who served as one of the expert consultants for the system, and for former classmates from those classes who acted as a test audience for the system.

The selection of the phrase models implemented resulted from identifying the various models introduced in early chapters of the text, and narrowing it down to a few which illustrated important features of the system. The random selection of models and vocabulary is illustrated merely by having enough different choices built in to the initial knowledge base. Other features are described in the sections below.

4.3.3. Vocabulary Selection

Vocabulary selection was fairly straightforward, once phrase models were selected. Each chapter introduces a finite set of vocabulary. Valid vocabulary choices for the different phrase models were compiled from each vocabulary set. The knowledge level associated with an exercise becomes the higher of the phrase model (chapter in which the sentence form was introduced) and the highest level of any individual word (chapter in which that word was introduced). This allows older (lower level) vocabulary to be used in more advanced phrase models, and more advanced vocabulary to be used in the earlier phrase models.

4.3.4. Base Phrase List and Pool Compilation

This is an offshoot of the vocabulary selection process. Since some phrase models share common lengths and word types, some base phrases can be shared between models. Conceptually, each model could have a single list, but in some cases, only a portion of that list is valid for another phrase model. The added layer of the pool supports all of the choices for a given model, and the list is the mechanism for sharing subsets of choices between models. A given base phrase should only appear in a single list. If a situation arises, where a base phrase appears to be needed in more than one list, this indicates the need to break up the lists.

4.3.5. Diagnosis Tree Design

This is done on a per–phrase model basis. The types of errors to be diagnosed for the given model are identified first. These come from the topics covered in the text where this phrase form is introduced, from previous concepts introduced, from predicting potential incorrect responses, and will be verified and enhanced by run–time information gathering. An example individual diagnosis tree is described in section 4.3.5.2.

4.3.5.1. Documentation Convention

As notational shorthand, several pieces of information are represented in the "Errors Diagnosed" section below. The emboldened lines indicate the specific concepts which
are tested by the given model. The indentation represents the sequential nature of the diagnosis, using the binary tree. For example, in section 4.3.5.2.2. below, it can be seen that it is first established that something is wrong with the first word, then established whether or not it is a gender-agreement problem. With some imagination, the binary tree can be visualized. Starting with any line, a left branch (the no-match path) can be taken by falling down to the next line which is at the same indentation level. A right branch (the match path) can be taken by going to the next contiguous line of increasing indentation. If there is no line with the same indentation level to fall to, the current line represents a left leaf node, and if there is no contiguous line of greater indentation, the current line represents a right node. The path followed on the way to an emboldened line indicates the diagnosis of a particular concept. The path which can be followed from an emboldened line, if any, indicates further diagnosis which the system is capable of (typically for giving the user hints).

4.3.5.2. Example Diagnosis Tree

4.3.5.2.1. Form of Model

The phrase represented by this model is a complete sentence taking the form "direct article, singular noun, present tense verb, adjective".

Knowledge level of model – 2.

Example phrase – The man is good.

Expected response – Der Mann ist gut.
4.3.5.2.2. Error Tree

- First word is different from expected
- Gender of article does not agree with noun
  - Gender variation 1 (see section 4.4.2.2.)
  - Gender variation 2
- Expecting a direct article
- Second word is different from expected
- Noun is not capitalized
- Synonym of noun used
- Third word is different from expected
- Tense of verb is incorrect
- Synonym of verb used
- Fourth word is different from expected
- Synonym of adjective used
- First and second words different from expected
- Synonym of noun used

4.3.5.2.3. Notes Concerning This Model

Note that use of a synonym for the noun appears twice in the diagnosis hierarchy. This is because an incorrect (or synonymous) noun may be of a different gender, which might also cause the article to be different from that expected. The types of errors initially diagnosed are taken from the text. Feedback from the system’s runtime gathered data may lead to enhancements of the tree.

4.3.6. Degree of Knowledge Base Design Goal Satisfaction

The initial knowledge base, illustrated by the sample system runs in the thesis report, supports all of the proposed features of the system. It provides a base for information gathering, and future enhancement should merely involve more of same process, with increasing numbers of selection choices.

4.4. Implementation

The system was implemented in C++. Prolog and C were used for earlier prototypes, and a variety of logic, pattern matching and object-oriented languages were considered for the actual thesis project. A language which supported object-oriented constructs was selected in order to ease the implementation of the object-oriented design. Without arguing the relative merits of the various object-oriented languages (or, for that matter, the merits of "pure" object-oriented languages versus hybrid languages), it is
safe to observe that C++ is becoming the defacto standard for commercial object-oriented implementations. It's availability on a wide variety of platforms was a major factor in Cadre Technologies decision to choose C++ for their object-oriented re-implementation of the popular Teamwork CASE tool [Wybolt-1990].

The tutoring system was implemented on Sun workstations, running SunOs v4.1, and compiled using Sun C++ v2.1.

4.4.1. Implementation Goals and Methodology

The primary goal of the implementation was to produce a functional system which displayed the promised functionality, without deviating drastically from the design as documented here. The method was an incremental approach. The Session agent was implemented as an empty driver only. Establishing a connection with the user, doing exercises, and logging run-time data were all stubs. Objects were coded one at a time, top-down, and placed in the system with the necessary stubs to allow it to be tested. As knowledge base objects were added, they were hard-coded with the most primitive amount of data to allow it to be used by the rest of the system. A major milestone in the implementation was the point at which the entire functional path of the system could be executed – login, exercise, diagnosis, data logging, logout. At that point the effort turned mainly to the addition of more and different knowledge.

4.4.2. Implementation Overview

One important detail that was deferred until the late design and early implementation phase was the actual method to be used for constructing the dynamic patterns needed for the binary diagnosis tree. In the analysis and design phases, this was represented by the PatternConstructor class/object, without specifying a great deal of detail about how the different and specific pattern needs would be handled. Early in the implementation phase, this was an object, but it quickly became apparent that it did not really meet the definition of an object. An object generally has a state, some data, and some things that it can do to/with that data. With the PatternConstructor, there was no clear way to have both data and a function. The pattern construction rules were not generic enough to permit them to be member data, used by a generic pattern building member function. The second approach was to make the PatternConstructors member functions of the PatternNode. Each Node would then know how to make the pattern it needed. This would work fine, but it was also true that certain Nodes, and even different Phrase-Models, had a need for the same constructor in different places. Finally, then, the PatternConstructors became utility functions. Each is implemented standalone, and PatternNodes maintain a pointer to the PatternConstructor function which fits its need. The function names are all prefixed with PC_ to indicate their use.

Besides just the manifestation of the PatternConstructor objects or functions, it was also a late decision on how to actually implement the construction. This was a potentially
messy programming area, and it was fundamental to the pattern masking diagnosis technique to be able to abstract this. The solution involved using the standard library function regexp(), and implementing pattern constructors which built patterns which were regular expressions. The actual pattern constructors are surprisingly small, most taking no more than a few lines of code. Two examples are included here to illustrate the implementation.

4.4.2.1. Pattern Constructor – Unexpected Word

Various static pattern fragments or regular expressions are defined externally to be used by phrase constructors, as needed. This constructor has a need for a match—a—single—word wildcard, so it uses the following:

```c
typedef char * pattern_fragment;

pattern_fragment any_word = "[a-zA-Z][a-zA-Z]*";
```

The following is the complete constructor function which generates the pattern to determine if the second word of a phrase is different from what was expected.

```c
Pattern PC_unexpected_second_word(Phrase & in_phrase)
{
    Pattern new_pattern = in_phrase;
    Word new_word;

    new_word.set_string(any_word);
    new_pattern.set_word(2, new_word);    // First argument is word
                                          // position in Phrase.

    return(new_pattern);
}
```

The creation of new_pattern works, because the Pattern object has a constructor which tells it how to make a Pattern from a Phrase. The pattern created is a regular expression which can be matched against the student response.

4.4.2.2. Pattern Constructor – Gender Variation

This example does not use any pre—defined pattern fragments. It illustrates a pattern constructor which uses the dictionary to create a variation of the expected phrase. This type of pattern is typically used in the tree when it is trying to isolate or diagnosis a potential anticipated error.

```c
Pattern PC_first_gender_var_first_word(Phrase & in_phrase)
{
    Pattern new_pattern = in_phrase;
```
Word old_word = in_phrase.get_word(1);  // Argument is word
// position in Phrase.

WordDescriptor new_word_WD(old_word);
int word_identifier = old_word.get_index();
Gender new_gender = new_word_WD.get_gender();
new_gender = (Gender) (((int) new_gender) % (G_none-1) + 1);
new_word_WD.set_gender(new_gender);
Word new_word = dictionary.get_word( word_index, new_word_WD );
new_pattern.set_word(1,new_word);
return(new_pattern);
}

This function could no doubt use a bit more explanation than the previous, but really it is not doing anything too difficult. This is one of the most complex pattern constructors in the system currently.

As in the previous example, the initial pattern is merely the input Phrase. The function then gets the current first word from the input Phrase. A WordDescriptor is created from this word (the WordDescriptor containing the feature attributes of the word), and the integer word identifier is obtained. (A word identifier and a WordDescriptor are the two things necessary for getting words from the dictionary. The final piece of setup involves getting the current Gender setting from the WordDescriptor. The Gender, and most attributes, are simply enumerated constants which represent the various values that the attribute can hold. The Gender definition looks like:

enum Gender
{
   G_DYNAMIC,
   G_masculine,
   G_feminine,
   G_neuter,
   G_none
};

The next cryptic-looking line in the example function is just calculating, what is internally known as, the first gender variation. It amounts to the next gender setting modulo the valid value. If the input gender had been masculine, the new gender would be feminine. If the input gender had been neuter, the new gender would be masculine. This new gender attribute is then set in the WordDescriptor, and the dictionary is called on to supply the new derived form of the indicated word. This new word is set in the pattern, and the pattern is complete. If the input phrase is "Der Mann..." the output pattern would be "Die Mann...".
5. CONCLUSIONS

5.1. Problems Encountered and Solved

Some implementation and portability problems were experienced. The standard library regular expression functions (regexp(), etc.), work as documented under Sun C++ v2.1, but do not compile properly under AT&T C++ v2.0. This was resolved in the simplest way – Sun C++ was used instead of AT&T C++. Problems were also encountered with the standard library time() function. This is used to seed the pseudo-random number generator for selecting phrase models and vocabulary. Under certain circumstances, invoking time would cause a segmentation fault. This turned out to be just a user error of passing an uninitialized pointer to the function, illustrating that it is just as easy to have pointer problems in C++ as it was in C. A final implementation problem involved getting dynamic binding to work. This is done with a compiler switch, but because of my date-dependent build procedures I was attempting to use the feature when not all modules had been compiled for it.

One problem which was wrestled with off-line was the issue of representing modified vowels, for example the u-umlaut in ”grün”. This problem is obviously not unique to translation programs. It is also an issue in areas where German is the first language, in relation to computers and various print media. For computers that support ”compose character”, and display screens that support special characters, the actual modified vowel could be displayed. However, ”compose character” is not a friendly operation for the user who must type it. There are three alternative conventions which can be used. The first approximates the umlaut by preceding the modified vowel with a double quote, e.g. gr"un. The second approximates the umlaut as if it was rotated 90 degrees clockwise, by following the vowel with a colon, e.g. gru:n. The third, and most generally accepted convention, is to append an ’e’ to the modified vowel, e.g. gruen. This convention was chosen, but in demonstrating the system, and explaining the problem, a number of users expressed a preference for the second convention. This issue might bear revisiting in a future version (see section 5.3.1., below).

5.2. Discrepancies and Shortcomings of the System

The system demonstrates most of the functionality described earlier in this report. One notable discrepancy is that, currently, the system only translates in one direction. It presents phrases in English, and reads and diagnoses German phrases. It is not believed that any major concepts were missed by implementing only a single direction, in fact, since the system attempts to diagnose German phrases, it must deal with language attributes, such as nouns with gender, which it would not have had to do in the opposite direction.

The user interface is mentioned below as an area where future enhancement is possible. It is worth noting as a discrepancy, also, that the current user interface is a bit fragile.
Unexpected input, such as escape sequences, will crash the system. A control-C sequence will abort the program. It would be desirable to prevent these situations so that the current run-time data log is not lost. From a user-friendliness standpoint, the user-interface exhibits a few other undesirable actions. After a bad menu selection, the menu is not re-displayed for the user. The '?' for "help" is not implemented at the phrase prompt – the user must answer incorrectly in order to see a menu of options. Finally, when asking for subsequent hints after an incorrect response, it is possible for the user's response to scroll off the screen (making it impossible for the user to refer back to it). None of these problems would be difficult to fix, but they were dropped as lower priority issues when time got tight.

Earlier in the report, synonyms were mentioned in the context of a translation tree. It was originally intended that the system would support some degree of alternative translation. Fundamental to this would be the ability of the system to handle synonymous words. This capability, however, did not make it into the prototype system. One means for handling this, would be to have the Word object maintain a data element which indicates synonyms (e.g. via word indices).

Several of the report-generation and file manipulation options shown in the system administration menu in the Sample Runs section are stubbed. The log file, however, is created, and the "view" options function as illustrated.

Additional capability needs to be added to the current DictionaryEntry types. The verb, for example, is basically stubbed. It would need to support additional tenses and variations, in a similar fashion to the current article and noun objects. Also, certain word types are not implemented at all, e.g. the adverb.

The final shortcoming which would prevent this system from truly being an effective tutoring aid, for a class such as the one on which it was based, is its depth of knowledge. It requires the expansion of its knowledge base to cover at least a full semester's worth of vocabulary and sentence forms. This increase in scope and volume would be necessary to give the tool some longevity with the students, as well as to prevent the repetition of exercises too quickly.

5.3. Suggestions for Future Work

The following subsections describe other less-essential enhancements, as well as possible project offshoots.

5.3.1. Potential Enhancements

Incorporating the basic enhancements described in section 5.2. would make the system useful for a particular course with a particular textbook. It would be nice, though, instead of hard-coded patterns, vocabulary and knowledge level, if a teacher would be able to customize the system to some extent (ideally, with-
out re--compilation). One of the easiest variations of this capability would be to allow the teacher to set the knowledge level for the vocabulary and sentence forms. This would allow the system’s behavior to be adjusted to a different textbook, or to a different resolution of knowledge level (e.g. sections within chapters, instead of whole chapters). A more ambitious variation of this capability would be to allow the teacher to define new sentence forms, choose vocabulary, and “program” the diagnosis by entering or modifying diagnosis trees. These capabilities could be implemented as a new type of user. Besides ”student” and ”system administrator”, there might then also be ”teacher”.

Neither the documentation conventions in this report, nor the actual source code, are particularly user–friendly ways to view and study the diagnosis trees. A feature which would be useful for future development, and mandatory for adding teacher–configuration features, is a pattern tree browser. This would allow an interactive user to display and traverse the various pattern trees which the system supports internally.

A possible enhancement to improve the system’s value to students, might be to examine including some lessons which relate to the concepts being tested/diagnosed by the system. If a student has trouble with an exercise, the system could offer a lesson on the concept involved (e.g. gender agreement) in addition to its current options of a hint or the correct answer.

The user interface for the current prototype is neither flashy, nor robust. This is an area where the system could be improved. The approach here could take two separate tacks, the easier of which being just a flashier front–end to make the system appear more professional or marketable. A more substantial direction, though, would be to research ways in which the user interface can be improved in order to make the system more effective. This could encompass not only the look and feel of the system, but also the manner in which exercises are presented, the timing of presentation, and other factor’s which would aid student comprehension.

Increased depth of knowledge of the current language was mentioned as a basic enhancement. This could also be a topic of advanced research, though. The present system was specifically intended as a prototype, and specifically targeted at the beginning student. It remains to be seen to what level the approach could be taken, or if it would fail to work for the more esoteric language issues encountered.

If the current prototype is to be expanded, it might bear examining the availability of other regular expression packages (besides the UNIX version which is currently used). The main shortcoming of the current package is that it does not support alternation (the ”or” function, or ‘!’). This would help tremendously to improve the ease with which different diagnosis nodes could be incorporated.

The system does not currently attempt to adapt its behavior based on a student’s history (previous executions of the system, or earlier exercises in the current
session). It may be possible for the system to draw some conclusions based on its diagnosis, for example if a certain type of error is made frequently. The system could then offer lessons, or choose exercise which will help to illustrate and reinforce the concept.

As mentioned earlier, there was a bit of an issue around the system’s representation of modified vowels (alternatives to using the umlaut). Three alternative conventions were presented in section 5.1. It is not clear that changing the current approach would be an enhancement, but what might be would be the ability for the teacher to choose the convention which best suits the class. (This may not be a light undertaking, though, since it does have knowledge base and translation implications).

5.3.2. Related Projects

This project, or selected concepts employed, could be applied to similar tutoring systems to support other languages. The current system could be expanded to support multiple languages, but it is not clear that the amount of commonality shared would justify the added size, complexity, and potential run time impact of combining them. Perhaps a better way to tie them together, if desired, would be through the use of a common front-end interface program. It is anticipated that that the concepts of this system would translate fairly well to other western (i.e. right-branching) languages, but its applicability to eastern (i.e. left-branching) languages would need to be examined in more detail.

The current system is fairly self-contained. While this works for the current scope, or even for an enhanced system with a complete semester’s worth of material, it does not lend itself as well to some of the more advanced enhancements suggested above. An interesting project would be to take the current system for its core functionality, and attempt to integrate it with external components, such as a commercial dictionary, morphological processor, etc.

A final related project might be to incorporate this system into a larger, multi-exercise system, or adding additional types of exercises to this system. This might include expanding (or narrowing) the current type of exercise to produce others. For instance, an easy variation would be the one-word phrase, amounting to a vocabulary quiz exercise. Alternatively, the phrase translation exercise could become one of multiple choices of types of exercises available.
6. GLOSSARY

Some of the terms below, related to learning/teaching/tutoring with computers, are used interchangeably by some authors. The definitions may be reversed by others. In older research works, certain terms may appear because others had not yet been introduced. The definitions below reflect my usage, and will be consistent with most current writings in this area.


Computer-Assisted (or Aided) Instruction (CAI) – This typically refers to the use of computers and educational software as learning aids by students (See also CAL and ITS).

Computer-Assisted (or Aided) Language Instruction (CALI) – A subset of CAI, specific to foreign language subjects.

Computer-Assisted (or Aided) Language Learning (CALL) – A subset of CAL, specific to foreign language subjects.

Computer-Assisted (or Aided) Learning (CAL) – This is the broadest category. In addition to the use of computers and educational software as learning aids by students (CAI), it includes the use of computers, educational software, and general-purpose software by teachers as tools for class management, testing and/or preparation of materials for students.

Drill-and-Practice – A mode of behavior for a CAI system in which it operates like electronic flashcards. The system presents a problem, the student answers, the system presents the next problem, etc.

Expert System – "... an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. Knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practitioners of the field.” [Feigenbaum–1985]


ITS – See Intelligent Tutoring System.

Intelligent Computer-Assisted (or Aided) Instruction (ICAI) – This term is a bit blurry, and is not commonly used anymore. It implies student-oriented software that exhibits some degree of intelligence. It would presumably fit somewhere between CAI and ITS.

Intelligent Tutoring System (ITS) – A CAI system which in some way attempts to provide intelligent diagnostics or lessons based on student input.
7. BIBLIOGRAPHY


Jackson, Eugene and Adolph Geiger (1985), German Made Simple, Doubleday, textbook for St. John Fisher College's Beginning German I and II courses.


Wirfs–Brock, Rebecca, Brian Wilkerson, and Lauren Wiener (1990), *Designing Object–Oriented Software*, Prentice Hall.


A. APPENDIX – USER MANUAL

A.1. Introduction

This manual describes the use of the German Language Tutoring system as a learning and practice aid. The system allows a student to practice phrase and sentence translation, and to receive some degree of assistance if needed. There are two categories of users, the "student" and the "system administrator". This manual describes the features available to the "student" user. "system administrator" features are described in a separate System Administration Manual.

The tutoring system currently supports a single type of exercise – phrase translation. After logging in (see section A.3.), the system will automatically begin presenting phrases and sentences to be translated.

A.2. Input/Output and Documentation Conventions

A.2.1. Help

The most important command to remember, when working with the system, is "?". Typing a question mark (followed by the "Enter" or "Return" key), at anytime that the system is waiting for input, will cause it to print instructions about what is expected.

A.2.2. Examples in this Document

In examples here, anything typed by the computer system, will be shown in bold, and anything that would have been typed by the user will be shown in italics. For example:

System typed this> user responded with this

A.2.3. Enter/Carriage-Return

As with many software packages, the German Language Tutor requires that the user terminate any line (username, phrase, or menu command number) with the "Enter" or "Return" key. This signals to the computer that it should read what was typed, and interpret it. Before hitting "Enter" or "Return", the line may be changed by deleting ("Delete" or "BackSpace" key) and re–typing characters. The "Enter" or "Return" action is often represented in documents as <cr> (short for carriage–return), so that the above example might have been shown as:

System typed this> user responded with this<cr>

However, this looks cluttered after a while, so it should just be assumed that it is required at the end of everything typed.

A.3. Getting Started

A.3.1. The Username

The system only allows validated users to execute it. To become validated, you must obtain a username from the designated system administrator. The "account" that you
get will be designated as "student" or "system administrator". If a user wishes to have both capabilities, two separate usernames must be obtained. The rest of this document assumes a "student" account.

A.3.2. Logging In

To start the tutoring system, type the command 'git' (short for German Language Tutor). The system should display its start-up message and version number. It will then ask for a chapter number. The user should enter the highest chapter number completed in the text. This will prevent the system from using vocabulary or concepts which have not yet been taught/learned. If the "current level" (saved from a previous login) is correct, the "Enter" or "Return key alone may be hit to remain at the same level. The following example illustrates a valid login by the user with the username "ken":

```
sys27> git
German Language Tutor, version 1.0

If you are not sure how to begin, type "?".

Please login: ken

Enter highest chapter completed (currently 2)? 3

Exercises will be limited to chapter 3 and earlier.

You will now be asked to translate various phrases and sentences.

Note that a "?" may be entered if this manual is not nearby, and the user needs refresher instructions. If the system says that the username is incorrect, it should be checked to see that it was typed correctly. It should be noted that the system is case-sensitive. That means that "a" is not the same as "A". Usernames will typically be all lower-case. If an incorrect username is entered three times, the system will give up and exit. The user should check with the system administrator to determine what his or her correct username is.

A.4. Doing Exercises

Once successfully logged in to the system, the translation exercises are automatically entered. A phrase or sentence will be presented, and the system will wait for a translation. The following example shows a correct response:

```
Translate: the water is fresh
```
to German:

*das Wasser ist frisch*

Yes! That is the expected response.

If the response does not match what the system was expecting (and note, that this does not necessarily mean that the translation is incorrect...), a list of options will be displayed:

That is not the expected translation. Enter the number indicating the action you would like to take:

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice?

The system will wait for the user to enter the number of the desired choice. Different lists of options may appear depending on the current scenario (for instance, if the system runs out of hints, selection 2 will not appear), but the above example shows most of the options, and they are further described below.

A.4.1. Trying Again

If the option of trying again is chosen, the system will offer the same phrase, and wait for another translation attempt. A user may choose to do this if he or she realizes what their initial mistake was, or to see if the correct answer can be reached by a little trial-and-error.

A.4.2. Receiving a Hint

Choosing to have the system give a hint will cause the system to attempt further diagnosis of the response. It will attempt to give a clue about what was wrong, without revealing the expected answer. Hints can be asked for repeatedly, and the system will attempt to supply more and more specific clues. When the system runs out of hints for a given response, receiving further hints will cease to be an option. The following example illustrates this process:

Translate:

*the table is round*

to German:

*die Tisch ist rund*

That is not the expected translation. Enter the number indicating the action you would like to take:
1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

There is something wrong with the first word in your response.

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

The gender of the first word is wrong.

1 - Try again.
2 - Have the system give you a hint.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice? 2

The system cannot diagnose this any further.

1 - Try again.
3 - Have the system give you its answer.
4 - Quit this exercise and go on to the next.
5 - Exit the tutoring system.
6 - Help.

Choice?

A.4.3. Receiving the Answer

If the user selects this option, the system will print the translation that it was expecting, and then present a new phrase for translation.
A.4.4. Quitting an Exercise

Quitting from an exercise amounts to the same thing as receiving the answer – the expected response is printed, and the system moves on to the next exercise.

A.4.5. Exiting the System

Exiting from the system causes the program to print the expected response for the current exercise, print some summary statistics, and terminate, leaving the user back at the computer’s command prompt.

A.4.6. Getting Help

Asking for help prints a brief summary that should clarify what the system is currently waiting for. Recall, also, that a ”?” at anytime brings the same thing.

A.5. Statistics

When running a question–and–answer system like this, it can be easy to lose track of time. As a reminder, and to break the monotony periodically, the system will interrupt the exercise sequence to present some statistics. The statistics will reflect the current session. The following example illustrates the type of information which is presented.

- Duration of session, so far – 20 minutes
- Number of different phrases asked to translate – 50
- Number of correct responses – 44
- Number of total responses – 57

Enter the number indicating the action you would like to take:

1 - Continue (return to exercises)
2 - Exit the tutoring system
6 - Help

Choice? 1

Translate:
.
.
.

Enter the number indicating the action you would like to take:
B. APPENDIX - SYSTEM ADMINISTRATION MANUAL

B.1. Introduction

This manual describes the German Language Tutoring system's administration. A login "username" must be obtained in order to use the system. It is the system administrator's responsibility to add and delete these usernames as required. Currently, a username must be identified as one of two types – a "student" account or an "administrator" account. A "student" account allows the system to be run for its intended purpose – to practice German. The features available to a "student" user are described in a separate User Manual.

An "administrator" account is not actually required for day-to-day administration of the system. With it, the user gains the ability to look at and manipulate the data gathered by the system at run-time. This data includes logs and statistics on student usage. This information is of particular interest to the designer who wishes to enhance the system, or fix specific shortcomings in diagnosis. This manual is presented in two main sections. Section B.2. discusses the day-to-day system administration – building from source code, adding users, etc. Section B.3. describes the features available to a user with an "administrator" account.

B.2. Off-Line System Administration

This section describes the source code directory structure, the build procedures, the resulting executables and support scripts, and the maintenance of usernames.


The tutoring system source code resides in a directory tree, beginning with the root directory 'glt' (short for German Language Tutor). The location of this tree may vary across systems. UNIX pages for 'CC', 'Id', 'make' and 'sccs', plus knowledge of csh scripts and the C and C++ languages may be of additional help in building and maintaining the system, depending on the level of understanding needed. The following picture illustrates the general directory tree structure.
There is no directory named "component_name". The dotted box above indicates the sub-tree structure which is repeated for multiple "components". A component is a logical grouping of source code and include files, and the directory name will reflect the logical function of the subdirectory (e.g. /models is the component directory which contains the system's PhraseModels). The /glt level can be referred to as the top level, and the /component level can be referred to as the component level (e.g. there is one top-level makefile, and multiple component-level makefiles).

'sccs' (UNIX Source Code Control System) is used for configuration management. All source code and build procedures are checked in to sccs, and should be checked out in order to make any modifications.

'make' (UNIX date-dependency, conditional build control program) is used to build the system. 'make' automatically looks for a file called 'makefile' to operate from. Simply giving the command 'make' in the top-level directory will automatically re-build whatever portion of the system has been modified since the last time it was built. 'make' can also be invoked at any component level, to build that component (compilation only) without linking it into the system.

The following sections describe the individual directories.

B.2.1.1. Component-Level Directories - /component_name

As mentioned above, a component is a logical grouping of source code. The current components of the tutoring system are listed below. A component is made up of some number of C++ source code files (.C), some number of include files (.h), and a makefile. All of these are checked into sccs (which maintains the /SCCS sub-directory). Everything else found under the component directory is dynamically built.

After a component is built, reference (read-only) copies of the .C files are left in the /src directory, and reference copies of the .h files are left in the /hdr directory. A reference copy of the makefile is left in the component-level directory. The out-
put of a component build is a single object module (named component\_name.o). It is left in the component-level directory, and will be used by the top-level makefile to construct a complete system (see below). The /obj directory is filled with the intermediate object modules (one per source file), and the /lis directory contains the intermediate C code which was generated from the C++ code (also one per source file. These are for reference only, and may be deleted to conserve disk space).

**B.2.1.1.1. /construction Component**

This component contains the source code for objects which are primarily responsible for dynamic phrase construction – word and phrase descriptors, pattern constructors, phrases and words.

**B.2.1.1.2. /dictionary Component**

This component contains the dictionary and dictionary entry objects, as well as the dictionary data.

**B.2.1.1.3. /log Component**

This component contains the run-time data gathering log and log entry objects.

**B.2.1.1.4. /models Component**

This component contains the objects and data which support the phrase models and base phrases.

**B.2.1.1.5. /session Component**

This component contains the controlling objects and agents for the session, statistics gathering, and the user interface.

**B.2.1.1.6. /users Component**

This component contains the objects which support user information.

**B.2.1.2. Top Level Directory – ../../git**

This directory represents the whole system. Here will be a makefile which will build anything that is out-of-date (including down in component directories) resulting in a complete and up-to-date system executable. The makefile will automatically invoke each component-level makefile. No .C files are maintained at the top level, but it does support top-level (global) include files. Include files are maintained at the top level if they are used by more than one component. A change to a global include file causes all components to re-build, so it is advantageous to keep include files at the component level if possible.
As at the component level, the makefile, and global include files are checked into scss. In addition, any documentation, support, or utility scripts will be checked into scss. Reference copies of this documentation will be put in the /notes sub-directory, and executable copies of the scripts will be left in the top-level directory. After a top-level build, the system executables will also be left in the top-level directory (see section B.2.2. for additional detail about these).

B.2.2. Executables

Because this system is a prototype, and because most of its future potential involves further modification, all test and debug capabilities have been left available. Simply invoking 'make' (or 'make all') at the top-level will build both debug and non-debug executables.

B.2.2.1. Non-debug executable - 'glt'

The normal executable, 'glt' will be built when 'make' is specified at the top level. To build 'glt' exclusively, specify 'make nodebug'. 'glt' is built with all debug code and feedback removed, so it is the smallest and fastest executable. This should be the one made generally available to students. SPECIAL NOTE: currently the separate 'glt' and 'dglt' executables do not exist. A single executable, 'glt' is built, and it is actually the version described as 'dglt' in this manual.

B.2.2.2. Debug executable - 'dglt'

'dglt' contains all of the code and functionality of 'glt', plus a large amount of debug and informational code. It is essentially a verbose version of glt. When run directly, dglt will display this information to the screen as it runs. The volume of information makes this rather unmanageable, though, so the 'split' script (section B.2.2.2. below) may be used to make dglt more practical to use.

B.2.2.2.1. Debug output conventions

As mentioned, a large volume of information is dumped when running the debug executable. To make this somewhat easier to use, a few basic conventions have been adopted. Refer to the following excerpt of debug information which illustrates these. The conventions are summarized following the excerpt.

```
ENTRY BasePhrase::get_index
DEBUG returning the index at array location [1]
DEBUG which is 6

"Normal system output"
ENTRY Word Constructor 1
ENTRY (for automatics)
```
• Debug information is broken into two categories. Lines beginning with ENTRY indicate a function or member function entry point. Reading down the line of ENTRYs yields a call–trace of the execution. All other information is tagged with DEBUG, to distinguish it from both the ENTRY points and the normal system output.

• DEBUG output lines are preceded by three tabs (24 spaces) and ENTRY output lines are preceded by five tabs (40 spaces). This is to visually separate them from each other, and to move them away from normal, left–justified, system output.

• Constructor and Destructor entry points have four spaces between the ENTRY keyword and the object name. Other functions have a single space. This is another visual aid, helpful when it is desired to ignore constructors and destructors (as peripheral to the execution trace).

• For objects that have multiple constructors, such as the Word object in the above example, the constructors are numbered, and followed by a brief comment, so that it is obvious when reading the log, which constructor was used.

• Subsequent DEBUG statements which have increasing number of spaces after the DEBUG keyword, in multiples of four spaces, reflect continuation or nested comment information.

• In the source code, all DEBUG and ENTRY output is sent to the standard error output stream (cerr) as opposed to the standard output stream (cout).

• Also in the source code, all output (debug or not) is flushed at the end of each line (using endl). This is to keep the output log accurate, should the system hang up somewhere. (If output was not flushed, some output could be buffered when the system hangs, thereby not accurately reflecting the execution prior to the problem.)

B.2.2.2.2. Supporting scripts

The 'split' script is designed to make using the 'dglt' executable easier. 'dglt' outputs a large amount of debug information, but must also output and input normal user interaction information. Running 'dglt' directly results in a large volume of output dumped to the screen. If the 'split' script is invoked, it will run 'dglt', redirecting the standard error output stream to a file named debug.log. Output to the screen will appear normal (the same as 'glt') and the debug output can be examined after the execution is complete. SPECIAL NOTE: As mentioned earlier, only a single executable is currently supported. It is named 'glt', but it contains the debug information described here as 'dglt'. Because of this, the 'split' script currently references 'glt', not 'dglt'.

B.2.3. Sample Build Output

The following sample output is included to illustrate the build procedure. It represents the build which results from changing a single .C file (phrasemodel.C in the 'models'
component. If multiple components had been changed, or if a global include file had been changed, the build output would have been much larger.

```
SYS27> make debug
#******************************* Update global header files.
#******************************* Check each sub-system of tutor.
for i in construction dictionary log models session users ; do \ 
    cd $i ; make GBL_HDRS="../hdr/basephrase.h ../hdr/dictionary.h
    ../hdr/log_defs.h ../hdr/log.h ../hdr/phrase.h ../hdr/phrasedescriptor.h
    ../hdr/phrasemodellist.h ../hdr/statistics.h ../hdr/stdtype.h ../hdr/
    user_defs.h ../hdr/userinterface.h ../hdr/userlist.h ../hdr/word.h
    ../hdr/worddescriptor.h" ; cd .. ; \ 
done
'construction.o' is up to date.
'dictionary.o' is up to date.
'log.o' is up to date.
#******************************* Get 'models' source code file
od src; scs -d../ get phrasemodel.C; cd ..;
1.54
177 lines
No id keywords (cm7)
#******************************* Compile 'models' source file
od src; CC -DDEBUG +i -c -I../hdr -I../.. hdr phrasemodel.C ; \ 
    rm *..c ; \ 
    mv phrasemodel.o ../obj/phrasemodel.o; cd ..
#******************************* Build a single 'models' object module
ld -r obj/BPT_no_translation.o obj/basephrase.o obj/basephraselist.o obj/
    basephrasepool.o obj/basephasetrans.o obj/data_phrasemodellist.o obj/pat-
    tern.o obj/phrasemodel.o obj/phrasemodellist.o obj/translationtree.o -o
    models.o
'session.o' is up to date.
'user.o' is up to date.
#******************************* Build the executable - 'dglt'.
CC construction/construction.o dictionary/dictionary.o log/log.o models/mo-
    dels.o session/session.o users/users.o -o dglt
SYS27>
```

Specifying 'make debug' requests that the debug executable 'dglt' be built. The build procedure first attempts to update any global include files which have changed (none have in this case). It then loops through the various components (passing down the list of global include files so that their modification dates can be considered by the components) invoking the build procedure at that level. The 'construction', 'dictionary' and 'log' components indicate that they are up to date. When the build procedure reaches
the 'models' component, it finds the file phrasemodel.C to be out of date. The source file is extracted from source control, and compiled with the DEBUG switch. The component then builds its object module (models.o). Back at the top level, the 'session' and 'user' components indicate that they are up to date. Finally, the top level build procedure builds the new debug executable, 'dglt'.

B.2.4. Adding Users

The tutoring system determines whether or not a user is authorized to use the system, by checking a file called users.data in the top level directory. This file is under SCCS control. There is a problem with this, though, in that the system may update this file (for a student's current knowledge level), without going through SCCS. An appropriate procedure for updating this file manually, would be:

- rename the current users.data file to a temporary name.
- sccs edit users.data (to check the file out for change).
- rename the temporary file back to users.data (overwriting the old version).
- make the necessary manual changes (i.e. adding or deleting users).
- sccs delta users.data (to check this version into source control).
- sccs get users.data (for the tutoring system to use).

A username entry consists of four lines in the users.data file. The first line is a single number, indicating whether this user is a student (1) or system administrator (2). The second line is also a single number, indicating the user's current knowledge level. This should be zero originally. The tutoring system will update the student's current level as needed. The third line is a single word which will be the user's login name. It should contain no spaces, and should be all lower case. The fourth line is currently used to describe the user in more detail, such as his or her first name. Currently, this line should also have no spaces in it. A new student entry might look like:

```
1
0
tjones
Thomas_Jones
```

B.3. System Administrator User Features

A separate manual exists describing the student's use of the system, and that manual should be considered a prerequisite to this one (both from a user standpoint, and because the conventions apply to this account as well). This document describes specifically the extra features available with a system administrator username.

Whenever student features are run on the system, data is gathered and appended to a log file called 'glt_admin.log'. This file is in an ASCII, though not too readable, for-
mat. The system administrator login provides some basic log manipulation, viewing, and report generating features. Access to these features is via simple menus, like those used during normal student execution. The available features are described below.

B.3.1. Delete Log

This is just what the name implies – the 'glt_admin.log' file is deleted. This might be done to start with a clean baseline (e.g. because the system has been modified), or to discard data which has outlived its usefulness. An alternative to this, though, is the Save Log feature described below.

B.3.2. Save Log

This effectively clears the data log, by copying its contents to a new file (the system will ask for a new file name). This can be used when it is desirable to start a new log, without losing data that has been previously gathered.

B.3.3. View Diagnosis Failures

One point where the system logs data, is when it must give up on a diagnosis (i.e. attempting to interpret a student’s translation response.) It is desirable to examine these cases to identify user errors which should have been handled by the system. This will be one of the key ways to improve the system’s performance against its current knowledge base. This feature allows the user to step through each incident report. The following brief interchange with the system after selecting this option illustrates the information available.

Enter the number indicating the action you would like to take:

[...]
7 - View Diagnosis Failures
[...]

Choice? 7

Enter Earliest Date Desired (MM-DD-YY)? 1-1-93

Date - 03-Jan-93 Username - ken
Index of Phrase Model which was used - 5
Phrase which was to be translated - The table is round.
Expected response - Der Tisch ist rund.
Actual response - But the system can’t tell me what’s wrong with this!

Enter the number indicating the action you would like to take:
B.3.4. Summarize Diagnosis Failures

This feature is very similar to on-line viewing of diagnosis failures, except that the system requests a file name, and generates a report summarizing all of the failures since the specified start date.

B.3.5. View Statistics

Whenever a session with the system is completed, an entry containing statistics from this session is entered in the log. This information might be of interest in characterizing users, qualifying other data, or monitoring system usage. Like View Diagnosis Failures, this feature asks for a date, and begins displaying statistics entries made since that date. The following is an example of a statistics entry.

Enter Earliest Date Desired (MM-DD-YY)? 1-1-93

Date - 03-Jan-93       Username - ken
User knowledge level at time of session - 2
Duration of session - 2 minutes
Number of different phrases asked to translate - 5
Number of correct responses - 4
Number of total responses - 7
Phrase Model indices answered correctly first time - 7, 3, 1, 9
Phrase Model indices where hints were given - 6
Phrase Model indices where answer had to be given - 6

Enter the number indicating the action you would like to take:

1 - Continue
2 - Quit

Choice? 1

Date - ...
B.3.6. View Cumulative Statistics

This feature is very similar to View Statistics. The system requests a date, and then compiles all of the individual statistics entries which have occurred since that date. The summary looks like the following:

Summary of statistics 03-Jan-93 through [current date]
Number of sessions - 8
Average user knowledge level - 3
Spread of user knowledge levels - 1-4
Average duration of session - 15 minutes
Total system usage time - 2 hours 0 minutes
Total number of different phrases asked to translate - 100
Total number of correct responses - 82
Total Number of responses - 115

B.3.7. Summarize Statistics

This is another report-generation feature. The system requests a file name, and generates a report summarizing all of the statistics entries made since the specified start date. It also includes the cumulative statistics based on all entries included in the report.

B.3.8. Full Report

This generates a report containing everything that is contained in the current log. (Basically a compilation of all of the previously described report generation features.)

B.3.9. Help

As in the student user environment, "help" can be selected at any time to obtain more detailed instructions from the system.
C. APPENDIX – COMMERCIAL SOFTWARE FOR GERMAN

Many of the products listed here have "sister" products which support other languages. For instance, in addition to "German Achievement I", Microcomputer Workshops Courseware also markets "French Achievement I" and "Spanish Achievement I".

Dasher – for Apple II computers by Conduit. This program is a drill-and-practice program which allows the teacher to enter the material to be drilled. Pre-designed data sets are available as companions to specific textbooks. Sentences are presented for translation. The student response is compared letter-to-letter with the expected response, and echoed back to the student with dashes (hence "dasher") in the place of discrepancies. This occurs a specified number of times before the correct answer is given. The teacher sets the re-try level.

Deutsch Aktuell – for Apple II computers by EMC Publishing. This is a collection of exercises and lessons covering vocabulary, grammar, reading comprehension, communication skills and culture. The exercises are classic drill-and-practice. If a student answers incorrectly, the correct answer is displayed, and the program moves on. There is no intelligent diagnosis. This program has also been criticized for its distracting shifts between graphics and text on the screen [LaReau–1989].

Foreign Language Courseware – German Level I Vocabulary/German Level II Vocabulary – for Apple II computers by Learning Technology. This is another classic drill-and-practice program for vocabulary. There is no diagnostic feedback, but the program does keep statistics on each exercise, including wrong answers, for later review.

The Game Show Subject Diskette – Foreign Language Words – for IBM PCs, Apple II and Commodore computers by Advanced Ideas. This program is a twist on vocabulary drill-and-practice. It is run as a game, and the student competes for a score. A clue to a word is presented in German, and the student must guess the English word. The more clues needed for a specific word, the fewer points earned. Although the student is only responding with single vocabulary words, he/she must read and comprehend the phrase and sentence clues.

German I Vocabulary/German II Vocabulary – for Apple II computers by Learning Technology, Inc. These are classic drill-and-practice vocabulary programs. The student translates individual words. Incorrect answers get one retry, then the program moves on.

German Achievement I/German Achievement II – for IBM PCs and Apple II computers by Microcomputer Workshops Courseware. German Achievement I is a drill-and-practice program for vocabulary. An incomplete sentence is presented, along with multiple choices for possible completions. If the student answers correctly, the program moves on. If the student answers incorrectly, the correct answer is given, along with a canned explanation associated with the incorrect choice. German Achievement II uses the same drill-and-practice scheme, but grammatical concepts are reviewed as opposed to vocabulary.
German Computer Grammar I and II – for Apple II computers by Lingo Fun. These programs are highly rated drill-and-practice programs for grammatical concepts. Brief canned lessons on each concept are available.

German Hangperson – for Apple II computers by Learning Technology, Inc. This program plays the game "hangman".

German Practice – for Apple II computers by George Earl. This program plays the game "hangman", using sentences, instead of single words. The student still guesses letters for hints.

German Review Packet – for IBM PCs and Apple II computers by COMPress. This is a drill-and-practice program for German Grammar. Each set of exercises begins with a brief lesson on the specific concept to be tested. The student is then asked questions requiring short (one or two word) answers. If the student answers incorrectly, one retry is given before the program gives the correct answer.

German Word Order – for Apple II computers by Gessler Educational Software. This is a sentence construction exercise program. All sentences are canned. The program chooses a sentence, then prints the words in random order. The student must attempt to reconstruct the correct sentence.

Mutilingual Story Teller – for Apple II computers by Lingo Fun. This program allows the student to practice writing skills. There are no intelligent diagnostics, so teacher feedback is required to determine correctness. The student enters statements in a binary tree, so that the story will be generated in different ways depending on which branch the program takes during play-back.

Snooper Troops – German – for Apple II computers by Gessler Educational Software. This is a role-playing game. The student is the detective, and the program supplies clues to a crime. The student must comprehend the clues, eliminate suspects, and identify the guilty person.

Vocabulary on the Attack – for Apple II computers by Langenscheidt. This is a multiple-choice drill-and-practice program for vocabulary. The student is presented with a word, and must select the correct translation. The correct answer must be given before the program will move on.

Vocabulary Workshop – for Apple IIe computers by Sterling Swift Publishing Company. This is a game variation of vocabulary drill-and-practice. The student begins with some play money, and then places wagers on his/her confidence in the chosen response. The translation words are presented as multiple choices. If the student is very sure of the answer, more money can be won (or lost) than if the student indicates less confidence. The program gives the correct answer and moves on after one attempt.

Wie Sagt Man? – for Apple II computers by Lingo Fun. This is very similar to the German Practice program described above, except that the hangman is missing (I guess that makes it more like "Wheel of Fortune"). The student guesses letters which are filled into the sentence to be guessed. The student has wide control over the level of difficulty. Higher difficulty levels will not show blanks for individual letters, and may not insert every occurrence of a successfully guessed letter.
**Wortgeecht** - for Apple II computers by Gessler Educational Software. This is a vocabulary drill program which plays the game "word attack". The student selects the category of vocabulary to be used (e.g. "sports and recreation"). Besides "word attack" the student may also browse the vocabulary list, take a multiple choice quiz, or do sentence completion.
D. APPENDIX – SOURCE CODE LISTINGS

All files included in this appendix are listed below. Header files appear first, listed alphabetically, followed by .C files, also listed alphabetically, followed by build procedures and execution scripts. For information on which header files are top-level, and which components the remaining files belong to, refer to the makefiles near the end of the listings (or the banner line of any individual listing).

The Session agent (session.C) is the main() entry point of the program.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Other</th>
<th>Build Procedures/Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>basephrase.h</td>
<td>PC_defs.h</td>
<td></td>
</tr>
<tr>
<td>basephraselist.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>basephrasepool.h</td>
<td>dictionary_defs.h</td>
<td></td>
</tr>
<tr>
<td>compoundstring.h</td>
<td>dictionary_defs.h</td>
<td></td>
</tr>
<tr>
<td>dictionary.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dictionaryentry.h</td>
<td>log_defs.h</td>
<td>statistics_defs.h</td>
</tr>
<tr>
<td>dynamicworddescriptor.h</td>
<td></td>
<td>user_def.h</td>
</tr>
<tr>
<td>hint.h</td>
<td></td>
<td>word_descriptor_defs.h</td>
</tr>
<tr>
<td>log.h</td>
<td></td>
<td>BPT_drop_fourth_word.C</td>
</tr>
<tr>
<td>logentry.h</td>
<td></td>
<td>BPT_no_translation.C</td>
</tr>
<tr>
<td>patternnode.h</td>
<td></td>
<td>PC_extra_article_fourth.C</td>
</tr>
<tr>
<td>phrasedescriptor.h</td>
<td></td>
<td>PC_gender1_first_word.C</td>
</tr>
<tr>
<td>phrasemodel.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phrasemodellist.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>statistics.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationtree.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>user.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>userinterface.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>userlist.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>word.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worddescriptor.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worddescriptors.h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D-1
<table>
<thead>
<tr>
<th>Objects</th>
<th>Other</th>
<th>Build Procedures/Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>basephrase.C</td>
<td>data_PC_defs.C</td>
<td></td>
</tr>
<tr>
<td>basephraselist.C</td>
<td>data_dictionary.C</td>
<td></td>
</tr>
<tr>
<td>basephrasepool.C</td>
<td>data_phrasemodellist.C</td>
<td></td>
</tr>
<tr>
<td>compounds</td>
<td>exit_tutor.C</td>
<td></td>
</tr>
<tr>
<td>dictionary.C</td>
<td>session.C</td>
<td></td>
</tr>
<tr>
<td>dictionaryentry.C</td>
<td>sys_exit_tutor.C</td>
<td></td>
</tr>
<tr>
<td>dynamicworddescriptor.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hint.C</td>
<td>utility_gender1_var.C</td>
<td></td>
</tr>
<tr>
<td>log.C</td>
<td>utility_gender2_var.C</td>
<td></td>
</tr>
<tr>
<td>logentry.C</td>
<td>utility_not_capitalized.C</td>
<td></td>
</tr>
<tr>
<td>patternnode.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phrasedescriptor.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phrasemodel.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phrasemodelhst.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>statistics.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>translationtree.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>user.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>userinterface.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>userlist.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>word.C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worddescriptor.C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**READ_ME**
- top-level makefile
- construction makefile
- dictionary makefile
- log makefile
- models makefile
- session makefile
- users makefile
<table>
<thead>
<tr>
<th>Objects</th>
<th>Other</th>
<th>Build Procedures/Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>baseline_all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>checkin_all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>checkout_all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cleanup_all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>print_all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>split</td>
</tr>
<tr>
<td></td>
<td></td>
<td>users.data</td>
</tr>
</tbody>
</table>
#ifndef PC_DEFS_H
#define PC_DEFS_H

// This offset can be used for case conversion
// of ASCII characters. Adding it to an upper
// case character yields lower case, and
// subtracting it from a lower case character
// yields upper case.

#define CASE_CONVERSION_OFFSET 32

#endif
// File: basephrase.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines the BasePhrase class.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.

#define BASEPHRASE_H

class BasePhrase
{
    public:
        BasePhrase(int in_length, int index1=0, int index2=0, int index3=0,
                    int index4=0, int index5=0, int index6=0, int index7=0,
                    int index8=0, int index9=0, int index10=0);
        ~BasePhrase();
        BasePhrase(const BasePhrase &);
        int get_index(int position);
        int get_size();
        int get_level();
    private:
        int length;
        int knowledge_level;
        int * indices;
};
#endif
#ifndef BASEPHRASELIST_H
#define BASEPHRASELIST_H

#include "basephrase.h"
#include "stdtype.h"

class BasePhraseList
{
public:
    BasePhraseList(int list_size, BasePhrase * first_phrase);
    -BasePhraseList();
    BasePhrase * get_BasePhrase();
    boolean reset_selections(int in_know_level);
    boolean set_level(int in_know_level);

private:
    int size;
    boolean * selected;
    BasePhrase * * phrases;
};
#endif
// Filename: basephrasepool.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines the BasePhrasePool class.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
// Revisions By Reason
// 26-Apr-93  Ken Staffan  v1.0 release with final thesis report.

#ifndef BASEPHRASEPOOL_H
#define BASEPHRASEPOOL_H

#include "basephraselist.h"

class BasePhrasePool
{
public:
    BasePhrasePool(int pool_size, BasePhraseList * list1=0, BasePhraseList * list2=0, BasePhraseList * list3=0, BasePhraseList * list4=0, BasePhraseList * list5=0, BasePhraseList * list6=0, BasePhraseList * list7=0, BasePhraseList * list8=0, BasePhraseList * list9=0, BasePhraseList * list10=0);

    ~BasePhrasePool();
    BasePhrase * get_BasePhrase();
    bool reset_selections(int in_know_level);
    bool set_level(int in_know_level);

private:
    int size;
    bool * selected;
    BasePhraseList ** lists;
};

#endif
# CompoundString Class

```cpp
class CompoundString {
public:
    CompoundString();
    ~CompoundString();
    CompoundString& operator=(CompoundString & in_CS);
    void set_word(int position, Word_in_word);
    Word get_word(int position);
    void get_string(char * in_ptf);
    int size;
    Word ** words;
protected:
};
```

### Class Phrase

```cpp
class Phrase {
public:
    Phrase(int in_size);
    Phrase(const Phrase &);
    ~Phrase();
    Phrase& operator=(Phrase & in_phrase);
    #ifdef DEBUG
    void debug_print();
    #endif
    void print();
    void read();
    int get_size();
    private:
};
```
May 9 1993 19:12:29 hdr/dictionary.h

1 //---------------------------------------------
2 //
3 // Filename: dictionary.h
4 //
5 // Project: German Language Tutor, RIT Master's Thesis
6 //
7 // Purpose: This file defines the Dictionary class.
8 //
9 // Notes: The theory and design of this system have been documented
10 // in the accompanying thesis report.
11 //
12 // Revisions By Reason
13 //---------- -------
14 // 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
15 //
16 //---------------------------------------------
17 #ifndef DICTIONARY_H
18 #define DICTIONARY_H
19 #include "word.h"
20 #include "worddescriptor.h"
21 //---------------------------------------------
22 // These forward references are used so that
23 // dictionary.h can be a top-level include
24 // file, while dictionaryentry.h resides at
25 // the component level.
26 //---------------------------------------------
27 class DictionaryEntry_noun;
28 class DictionaryEntry_article;
29 class DictionaryEntry_adjective;
30 class DictionaryEntry_verb;
31 //---------------------------------------------
32 // The Dictionary class represents all of the
33 // words and morphological derivatives
34 // supported by the system.
35 //---------------------------------------------
36 class Dictionary
37 { public:
38     Dictionary(DictionaryEntry_noun * noun_list, 
39                    DictionaryEntry_article * article_list,
40                    DictionaryEntry_adjective * adj_list, 
41                    DictionaryEntry_verb * verb_list); 
42     ~Dictionary();
43     int get_level(int word_index);
44     Word get_word(int word_index, WordDescriptor * word_desc);
45 private:
46     DictionaryEntry_noun * nouns;
47     DictionaryEntry_article * articles;
48     DictionaryEntry_adjective * adjs;
49     DictionaryEntry_verb * verbs;
50     ~Dictionary();
51 }
52 #endif
// File name: dictionary_defs.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file defines symbols supporting various dictionary objects.
// Notes: The theory and design of this system have been documented in the accompanying thesis report.
// Revisions By Reason
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#ifndef DICTIONARY_DEFS_H
#define DICTIONARY_DEFS_H

#define DE_MIN_NOUN_INDEX 1
#define DE_MAX_NOUN_INDEX 99999
#define DE_MIN_DIRART_INDEX 100001
#define DE_MAX_DIRART_INDEX 199999
#define DE_MIN_ADJ_INDEX 200001
#define DE_MAX_ADJ_INDEX 299999
#define DE_MIN_VERB_INDEX 300001
#define DE_MAX_VERB_INDEX 399999

#endif
class DictionaryEntry_noun 
  public: DictionaryEntry
  public: DictionaryEntry_noun
  public: DictionaryEntry
  public: DictionaryEntry
  public: DictionaryEntry
  public: DictionaryEntry
  public: DictionaryEntry
  public: DictionaryEntry
  public: DictionaryEntry
  public: DictionaryEntry
// File: dynamicworddescriptor.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines the DynamicWordDescriptor class.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.

#ifndef DYNAMICWORDDESCRIPTOR_H
#define DYNAMICWORDDESCRIPTOR_H

#include "stdtype.h"
#include "word.h"

class DynamicWordDescriptor
{
  public:
    DynamicWordDescriptor();
    ~DynamicWordDescriptor();
    Count get_count();
    Gender get_gender();
    Language get_language();
    void resolve_dynamics(Word * in_word);

  private:
    Count count;
    Gender gender;
    Language language;
};

#endif
// Filename: hint.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines the Hint class.
// Notes: The theory and design of this system have been documented in the accompanying thesis report.

// Revisions


#define HINT_H

// The Hint class represents an individual clue associated with a particular pattern in a diagnosis tree.

class Hint
{
    public:
        Hint(char * text);
        ~Hint();
        void print();
    private:
        char * hint;
};

#endif
// File: log_defs.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file defines some log-related symbols.
// Notes: The theory and design of this system have been documented in the accompanying thesis report.

#define LOG_DECLS_H
#define LOG_DECLS_H

#define MAX_CURRENT_LOG_ENTRIES 1000

#define MAX_DATE_STAMP_SIZE 50

enum LE_type
{
    LE_UNKNOWN_TYPE,
    LE_at_leaf,
    LE_final_stats
};

enum LE_at_leaf_type
{
    LE_UNKNOWN_AT_LEAF_TYPE,
    LE_at_leaf_hint,
    LE_at_leaf_diag
};

};
May 9 1993 19:12:29 log/hdr/logentry.h Page 1

1 //
2 /***********/
3 /*
4 File: logentry.h
5 Project: German Language Tutor, RIT Master's Thesis
6 Purpose: This includes the LogEntry class, a single
7 entry in the run-time log, and its derived classes.
8 Notes: The theory and design of this system have been documented
9 in the accompanying thesis report.
10 Revisions By Reason
11 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
12 /***********/
13
14 ifndef LOGENTRY_H
15 #define LOGENTRY_H
16
17 #include <fstream.h>
18 #include <time.h>
19 #include "log_defs.h"
20 #include "compoundstring.h"
21 #include "statistics.h"
22
23 // The LogEntry class represents a single informational entry in the run-time log.
24 //
25 class LogEntry
26 {
27   public:
28     LogEntry();
29     LogEntry(char * in_username, int in_index, LE_type in_type);
30     ~LogEntry();
31     virtual void print(ofstream * log_stream)=0;
32     LE_type get_type();
33   protected:
34     time_t date_time_stamp;
35     char * username;
36     int model_index;
37     LE_type record_type;
38   private:
39   }
40
41 // This derived class represents the log entry
42 // when the diagnostic tree hits a leaf
43 // node and gives up.
44 class LogEntry_at_leaf : public LogEntry
45 {
46   public:
47     LogEntry_at_leaf(LE_type in_type);
48     LogEntry_at_leaf(char * in_username, int in_index, char *
49       in_preset_phrase, LE_type in_type, Phrase in_exp_response,
50       Phrase in_user_response);
51     ~LogEntry_at_leaf();
52     void print(ofstream * log_stream);
53     void read(ifstream * log_stream);
54     void display();
55   }
56
57 #endif

May 9 1993 19:12:29 log/hdr/logentry.h Page 2

67 private:
68     LE_at_leaf_type;
69     char * presentation_phrase;
70     char * expected_response;
71     char * user_response;
72 
73 // This derived class represents the log
74 // entry made at the completion of a session.
75 //
76 class LogEntry_final_stats: public LogEntry
77 {
78   public:
79     LogEntry_final_stats(LE_type in_type);
80     LogEntry_final_stats(char * in_username, Statistics in_stats);
81     ~LogEntry_final_stats();
82     void print(ofstream * log_stream);
83     void read(ifstream * log_istream);
84     void display();
85     Statistics * get_stats();
86     private:
87     Statistics current_stats;
88   }
89 
90 #endif
May 9 1993 19:12:29 models/hdr/patternnode.h Page 1

1 //**********************************************************************
2 // Filename: patternnode.h
3 // Project: German Language Tutor, RIT Master's Thesis
4 // Purpose: This include file defines the PatternNode class.
5 // Notes: The theory and design of this system have been documented
6 // in the accompanying thesis report.
7 // Revsions By Reason
8 // --- ----
9 // 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
10 //**********************************************************************
11 #ifndef PATTERNNODE_H
12 #define PATTERNNODE_H
13 #include "compoundstring.h"
14 #include "hint.h"
15 //---------------------------------------------------------------------
16 // The PatternNode class represents a single //
17 // node in a binary translation tree. //
18 //---------------------------------------------------------------------
19 class PatternNode
20 {
21 public:
22 PatternNode(Pattern *in_func, Phrase *in_mat, //
23 PatternNode * in_mat, Hint * in_hint); //
24 ~PatternNode();
25 boolean process_response(Phrase expected_res, Phrase user_res); //
26 boolean more_hints(Phrase expected_response, Phrase user_response); //
27 private:
28 Hint * hint;
29 Pattern *(pattern_constructor) (Phrase &); //
30 PatternNode * match_path_pattern;
31 PatternNode * no_match_path_pattern;
32 }; //
33 #endif
#ifndef PHRASEDESCRIPTOR_H
#define PHRASEDESCRIPTOR_H

#include "compoundstring.h"
#include "worddescriptor.h"
#include "basephrase.h"

class PhraseDescriptor
{
    public:
        PhraseDescriptor(int list_size, int * in_order, WordDescriptor * wd1,
                          WordDescriptor * wd2=0, WordDescriptor * wd3=0,
                          WordDescriptor * wd4=0, WordDescriptor * wd5=0,
                          WordDescriptor * wd6=0, WordDescriptor * wd7=0,
                          WordDescriptor * wd8=0, WordDescriptor * wd9=0,
                          WordDescriptor * wd10=0);
        ~PhraseDescriptor();
        Phrase build_phrase(BasePhrase* in_bp);
    private:
        int size;
        int * fillin_order;
        WordDescriptor ** descriptors;
};
#endif
#ifndef PHRASE_MODEL_H
#define PHRASE_MODEL_H

#include "phraselocator.h"
#include "translationenv.h"

class PhraseModel
{
    public:
        PhraseModel(int in_know_level,
                     BasePhrasePool* pool, PhraseDescriptor* in_desc,
                     TranslationTree* in_tree);
        ~PhraseModel();
        bool do_exercise();
        bool reset_depletions(int in_level);
        bool set_level(int in_level);
    private:
        int knowledge_level;
        BasePhrasePool* phrases;
        PhraseDescriptor* fixed_phrase_desc;
        TranslationTree* trans_tree;
};

#endif
---

FILENAME: phraseModelList.h

PROJECT: German Language Tutor, RIT Master's Thesis

PURPOSE: This include file defines the PhraseModelList class.

NOTES: The theory and design of this system have been documented in the accompanying thesis report.

---

 ifndef PHRASEMODELLIST_H
 define PHRASEMODELLIST_H

 include "stdtype.h"

 // This forward reference is used so that phraseModelList.h can exist as a top-level makefile, while phraseModel.h resides at the component level.

 class PhraseModel;

 // The PhraseModel class represents the collection of all PhraseModels which the system supports.

 class PhraseModelList
 |
 public:
 PhraseModelList (int in_size, PhraseModel * list);
 -PhraseModelList();
 void do_exercise();
 void reset_depletions();
 void set_current_knowledge_level (int in_level);

 private:
 int size;
 int current_level;
 boolean * depleted;
 PhraseModel * model_list;
 boolean find_and_try();


 endif
May 9 1993 19:12:29  session/hdr/session.h  Page 1

1 //-----------------------------------------------
2 // Filename: session.h
3 // Project: German Language Tutor, RIT Master's Thesis
4 // Purpose: This include file defines symbols used by the controlling
5 // agent "session"
6 // Notes: The theory and design of this system have been documented
7 // in the accompanying thesis report.
8 // Revisions By Reason
9 // ----------- ----
10 // 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
11 //-----------------------------------------------
12
13 #ifndef SESSION_H
14 #define SESSION_H
15
16 //------------------------------///
17 // monitoring, the major value should always be //
18 // a multiple of the minor value. //
19 //------------------------------///
20
21 #define MINOR_INTERVENE_MAX 50
22 #define MAJOR_INTERVENE_MAX (3 * MINOR_INTERVENE_MAX)
23
24 #endif
#ifndef STATISTICS_H
#define STATISTICS_H

#include <fstream.h>

// The Statistics class compiles run-time data //
// for the current user's benefit, and to be //
// be made available to the system //
// administrator. //

class Statistics
{
public:
    Statistics();
    ~Statistics();
    void increment_exercise_count();
    void increment_attempt_Count();
    void increment_correct_count();
    void print_summary();
    void print_to_file(ofstream * stats_stream);
    void read(istream * stats_stream);
    void display();
    int get_prev_duration();
    int get_exercise_count();
    int get_attempt_Count();
    int get_correct_count();
private:
    int prev_duration;
    int exercise_count;
    int attempt_Count;
    int correct_Count;
};
#endif
// File: statistics_defs.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines symbols used by the statistics class.
// Notes: The theory and design of this system have been documented in the accompanying thesis report.
// Revisions By Reason
// 26-Apr-93 Ken Gronf Anv 1.0 release with final thesis report.

#define STATISTICS_DECLS_H
#define STATISTICS_DECLS_H

// Date/time stamp definitions.

#define NUM_SECONDS_PER_MINUTE 60.0
#define NUM_SECONDS_PER_HOUR  (60.0 * NUM_SECONDS_PER_MINUTE)
#define NUM_SECONDS_PER_DAY  (24.0 * NUM_SECONDS_PER_HOUR)
#define MAX_SESSION_DURATION NUM_SECONDS_PER_DAY

#undef STATISTICS_DECLS_H
#ifndef STDTYPE_H
#define STDTYPE_H

// The pattern fragment type can take the
// place of Words as Patterns are constructed
// with embedded wildcards.

typedef char* pattern_fragment;

// A boolean data type and its values.

typedef int boolean;

#define TRUE 1
#define FALSE 0
#endif
#include <stdlib.h>
#include "phrasedescriptor.h"
#include "basephrase.h"
#include "patternnode.h"

// The TranslationTree class knows how to translate, and diagnose a particular phrase
// model.

class TranslationTree
{
  public:
    TranslationTree(BasePhrase *in_bpt) (BasePhrase &),
    PatternNode * in_root,
    PhraseDescriptor * in_fixed_desc);
    -TranslationTree();
    boolean check_response(BasePhrase in_phrase);
  private:
    BasePhrase (*in_phrase_translator) (BasePhrase &);
    PatternNode * root_pattern;
    PhraseDescriptor * fixed_phrase_desc;
};
May  9 1993 19:12:29  users/hdr/user.h  Page 1

1 //-----------------------------
2 // Filename: user.h
3 // Project: German Language Tutor, RIT Master's Thesis
4 // Purpose: This include file defines the User class - a single
5 // entry in the UserList of authorized users.
6 //
7 // Notes: The theory and design of this system have been documented
8 // in the accompanying thesis report.
9 //
10 // Revisions  By        Reason
11 // --------  ------
12 // 26-Apr-93  Ken Staffan  v1.0 release with final thesis report.
13 //
14 //
15 #ifndef USER_H
16 #define USER_H
17
18 #include "fstream.h"
19 #include "stdtype.h"
20 #include "user_defs.h"
21
22 //-------------------------------
23 // The User class represents a single valid  //
24 // user of the system.
25 //-------------------------------
26
class User
27 {
28 public:
29   User();
30   ~User();
31   bool read(ifstream * user_stream);
32   User type match_user(char * in_string);
33   char * get_string();
34   int get_current_level();
35   bool set_current_level(int in_level);
36   void write(Gfstream * user_stream);
37
38 private:
39   int last_known_level;
40   User type;
41   char * login_name;
42   char * full_name;
43};
44
45 #endif
#ifndef USER_DEFS_H
#define USER_DEFS_H

/*
 * Different login user types.
 */
enum Usertype
{
  NO_USERTYPE,
  STUDENT,
  SYSTEM,
};

/*
 * Maximum sizes for user-related strings.
 */
#define MAX_LOGIN_NAME_STRING 25
#define MAX_FULL_NAME_STRING 100

/*
 * Maximum number of different usernames supported.
 */
#define MAX_NUMBER_USERS 100

/*
 * Current range of supported knowledge level.
 */
#define MIN_KNOW_LEVEL 1
#define MAX_KNOW_LEVEL 6

#define USER_DEFS_H // These lines are removed in the actual implementation
// File name: userinterface.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This includes file defines the UserInterface class.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
// Revisions By Reason
// ------- -- ------
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#ifndef USERINTERFACE_H
#define USERINTERFACE_H

#include "hint.h"
#include "userinterface_defs.h"

// The UserInterface object allows for
// encapsulation of the actual U/I code. Could
// later be replaced with a "flashier"
// interface. 

class UserInterface
{
public:
  UserInterface();
  ~UserInterface();
  ControlCommand wrong_answer();
  ControlCommand after_hint(Hint * in_hint);
  ControlCommand cant_diagnose_further();
  ControlCommand after_stats();
  ControlCommand sysadmin();
  ControlCommand after_view();
  ControlCommand after_sum();
private:
};
#endif
    // File:   userinterface_defs.h
    // Project: German Language Tutor, RIT Master's Thesis
    // Purpose: This include file defines symbols related to the
    //           UserInterface class.
    // Notes: The theory and design of this system have been documented
    //        in the accompanying thesis report.

    // Revisions
    // -------
    // 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

    //------------------------------------------------------------------
    #ifndef USERINTERFACE_DEFS_H
    #define USERINTERFACE_DEFS_H

    //------------------------------------------------------------------
    // Controls enumerates all possible
    // menu selections. They may not all be valid
    // at a given prompt.
    //------------------------------------------------------------------

    enum ControlCommand {
        CC_NONE,
        CC_try_again,
        CC_hint,
        CC_correct_answer,
        CC_quit,
        CC_exit,
        CC_help,
        CC_admin_write,
        CC_admin_delete,
        CC_admin_save,
        CC_admin_diags,
        CC_admin_report_diags,
        CC_admin_stats,
        CC_admin_report_stats,
        CC_admin_cum_stats,
        CC_admin_report_cum_stats,
        CC_admin_report_full,
        CC_admin_find_next
    };

    #endif
// Filename: userlist.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines the UserList class.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
// Revisions By Reason
// -------------- ----------
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#ifndef USERLIST_H
#define USERLIST_H

#include "user_defs.h"
#include "stdtype.h"

class User;

class UserList
{
  public:
    UserList();
    ~UserList();
    UserType establish_connection();
    void read();
    int get_current_level();
    void write();

  private:
    int current_user;
    User ** users;
    int size;
    boolean list_modified;
};

#endif
#ifndef WORD_H
#define WORD_H

#include "stdtype.h"
#include "worddescriptor.h"
#include "word_and_phrase_defs.h"

class Word
{
public:
    Word(char * in_string, WD_Type in_type, Count in_count, Gender
        in_gender, Language in_lang); // Intended for knowledge base

    void init(); // Build.
    void Word(const Word &); // Intended for automatics.
    ~Word();
    Words operator=(Word & in_word);
    boolean read();
#endif DEBUG
    void debug_print();
#endif

    void print();
    void set_string(char *);
    void set_type(WD_Type in_type);
    void set_language(Language in_lang);
    void set_gender(Gender in_gend);
    void set_count(Count in_count);
    void set_index(int in_index);
    int get_index();
    WD_Type get_type();
    Count get_count();
    Gender get_gender();
    Language get_language();

    char * get_string();

private:
    char * word;
    int index;
    WD_Type type;
    Count count;
    Gender gender;
    Language language;
};
#endif
#define WORD_AND_PHRASE_DECLS
#define WORD_AND_PHRASE_DECLS_N

#define MAX_WORD_LENGTH 20

#define MAX_PHRASE_LENGTH 10
#define MAX_PHRASE_STRING_LENGTH (MAX_PHRASE_LENGTH * MAX_WORD_LENGTH)

enum Gender {
    G_DYNAMIC,
    G_masculine,
    G_feminine,
    G_neuter,
    G_none
};

enum Count {
    C_DYNAMIC,
    C_singular,
    C_plural,
    C_none
};

#define L_DYNAMIC,
#define L_english,
#define L_german,
#define L_none

#define 26-Apr-93 Ken Staffan v.1.0 release with final thesis report.
// File name: worddescriptor.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines the WordDescriptor class.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.

#ifndef WORDDESCRIPTOR_H
#define WORDDESCRIPTOR_H

#include "stdtype.h"
#include "worddescriptor_defs.h"
#include "word_and_phrase_defs.h"

class Word;

class DynamicWordDescriptor;

// The WordDescriptor class specifies
// the attributes of a derivation of a word.

class WordDescriptor
|
public:
  WordDescriptor(Count in_count, Gender in_gend, Language in_lang,
                  WD_Type in_type);
  WordDescriptor(Word & in_word);
  ~WordDescriptor();
  Count get_count();
  Gender get_gender();
  Language get_language();
  void set_count(Count in_count);
  void set_gender(Gender in_gend);
  void set_language(Language in_language);
  bool is_dynamic();
  void resolve_dynamics(DynamicWordDescriptor & in_dyna);
  WD_Type type;
private:
  Count count;
  Gender gender;
  Language language;
};

#endif
// File: worddescriptor_defs.h
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This include file defines symbols related to the
//           WordDescriptor class.
// Notes: The theory and design of this system have been documented
//        in the accompanying thesis report.

// Revisions By        Reason
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#ifndef WORDDESCRIPTOR_DEFS_H
#define WORDDESCRIPTOR_DEFS_H

// The various types of word descriptors.

enum WD_Type
{
    WD_UNKNOWN_TYPE,
    article,
    noun,
    verb,
    adjective
};
#endif
---

// File: BPT_drop_fourth_word.C

// Project: German Language Tutor, RIT Master's Thesis

// Purpose: This file implements the base_phrase_translator function of
generating a BasePhrase, appropriately modified to be correct
in the other language. In this case, the fourth word in
the basephrase is dropped.

// Notes: The theory and design of this system have been documented
in the accompanying thesis report.

// Revisions By Reason
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

// ----------------- Included files -----------------//

#include <iostream.h>
#include "stdtype.h"
#include "basephrase.h"

// Executable code begins here. //

BasePhrase BPT_drop_fourth_word(BasePhrase & in_basephrase)
{
    if (DEBUG
        cerr << " ENTRY BPT_drop_fourth_word() " << endl;
    endif

    int old_size = in_basephrase.get_size();
    BasePhrase new_bp( old_size -
        in_basephrase.get_index(1),
        in_basephrase.get_index(2),
        in_basephrase.get_index(3) );

    return(new_bp);
}
---
// Filename: BPT_no_translation.C
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file implements the base_phrase_translator function of
// generating a BasePhrase, appropriately modified to be correct
// in the other language. In this case, no modifications are
// needed.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
// Revisions By Reason
// -------- -- -------
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#include <iostream.h>
#include "stdtype.h"
#include "basephrase.h"

BasePhrase BPT_no_translation(BasePhrase & in_basephrase) {
  #ifdef DEBUG
cerr << "\t\tENTRY BPT_no_translation()" << endl;
  #endif
  return(in_basephrase);
}
Filename: PC_extra_article_fourth.C
Project: German Language Tutor, RIT Master's Thesis
Purpose: This file implements the pattern constructor function of generating a Pattern which will match the input Phrase, regardless of whether or not an extra article has been included in the fourth word position.

Notes: The theory and design of this system have been documented in the accompanying thesis report.

Revisions

By Reason
26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#include <iostream.h>
#include "stotype.h"
#include "compoundstring.h"

extern pattern_fragment any_word;

Pattern PC_extra_article_fourth(Phrase & in_phrase)
{
    //---------- Automatics - objects ----------
    Phrase bigger_phrase(in_phrase.get_size()+1);
    Word new_word;
    //---------- Automatics - other ----------
    //---------- Code begins ----------
    #ifdef DEBUG
    cerr << "\t\t\tENTRY PC_extra_article_fourth()" << endl;
    #endif
    new_word.set_string(any_word);
    bigger_phrase.set_word(1, in_phrase.get_word(1));
    bigger_phrase.set_word(2, in_phrase.get_word(2));
    bigger_phrase.set_word(3, in_phrase.get_word(3));
    bigger_phrase.set_word(4, new_word);
    bigger_phrase.set_word(5, in_phrase.get_word(4));
    Pattern wildcard_pattern(bigger_phrase);
    return wildcard_pattern;
}
construct.arc/src/PC_gender1_first_word.C  Page 1

---
// Filename: PC_gender1_first_word.C
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file implements the pattern constructor function of
// generating a Pattern which will match the expected Phrase,
// with the first gender variation of the first word.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
---

// ------------------ Included files ------------------
#include <iostream.h>
#include "stdtype.h"
#include "compoundstring.h"
#include "word.h"
#include "wordselector.h"
#include "dictionary.h"
---
// ------------------ External Data ------------------
extern Dictionary dictionary; // The dictionary will be used to
// obtain the gender variation of
// the needed word.
---
// ------------------ External functions ------------------
extern Gender utility_gender1_var(Gender);
---
// ------------------ Calling syntax ------------------
Pattern PC_gender1_first_word(Phrase & in_phrase)
{
    ---
    // ------------------ Automatics - objects ------------------
    Pattern new_pattern = in_phrase;
    Word old_word = in_phrase.get_word(1);
    WordDescriptor new_word WD(old_word);
    ---
    // ------------------ Automatics - other ------------------
    int word_index = old_word.get_index();
    Gender new_gender;
    ---
    // ------------------ Code begins ------------------
    #ifdef DEBUG
    cerr << "\t\tENTRY PC_gender1_first_word()" << endl;
    #endif
    ---
    // Obtain the first gender variation of the
    // current gender.
    ---
    new_gender = utility_gender1_var(new_word WD.get_gender());

construct.arc/src/PC_gender1_first_word.C  Page 2

---
// Set this attribute in the WordDescriptor for
// the new gender.
---
new_word WD.set_gender(new_gender);
---
// Obtain the new gender variation of the word
// from the dictionary, and set this as the
// first word in the generated pattern.
---
new_pattern.set_word(1, dictionary.get_word( word_index, &new_word WD ));
---
return( new_pattern );
---
1 // Filename: PC_gender2_first_word.C
2 // Project: German Language Tutor, RIT Master's Thesis
3 // Purpose: This file implements the pattern constructor function of
4 // generating a Pattern which will match the expected Phrase,
5 // with the second gender variation of the first word.
6 // Notes: The theory and design of this system have been documented
7 // in the accompanying thesis report.
8 // Revisions By Reason
9 // 24-Apr-93 Ken Staffan v1.0 release with final thesis report.
10 //
11 //------------------------------- Included files ------------------------------
12 //
13 #include <iostream.h>
14 #include "stdafx.h"
15 #include "compoundstring.h"
16 #include "word.h"
17 #include "worddescriptor.h"
18 #include "dictionary.h"
19 //------------------------------- External Data -------------------------------
20 extern Dictionary dictionary; // The dictionary will be used to
21 // obtain the gender variation of
22 // the needed word.
23 //------------------------------- External functions ----------------------------
24 extern Gender utility_gender2_var(Gender);
25 //------------------------------- Calling syntax ------------------------------
26 Pattern PC_gender2_first_word(Phrase & in_phrase)
27 {
28 //------- Automatics - objects ----------------------------------
29 Pattern new_pattern = in_phrase;
30 Word old_word = in_phrase.get_word[1];
31 WordDescriptor new_word_MD(old_word);
32 //------- Automatics - other ---------------------------------------
33 int word_index = old_word.get_index();
34 Gender new_gender;
35 //------- Code begins ----------------------------------------------
36 #ifdef DEBUG
37 cerr << "\t\t\tENTRY PC_gender2_first_word()" << endl;
38 #endif
39 // Convert the current gender
40 // attribute to its second variation.
41 new_gender = utility_gender2_var(new_word_MD.get_gender());
42 // Set this attribute in the WordDescriptor for
43 // the new word.
44 new_word_MD.set_gender(new_gender);
45 // Obtain the new gender variation of the word
46 // from the dictionary, and set this as the
47 // first word in the generated pattern.
48 new_pattern.set_word(1, dictionary.get_word(word_index, &new_word_MD));
49 return( new_pattern );
50 }
May 9

---

1 //
2 //
3 // File:      PC_noun_cap_second_word.c
4 //
5 // Project:  German Language Tutor, RIT Master's Thesis
6 //
7 // Purpose:  This file implements the pattern constructor function of
8 //
9 // Notes:     The theory and design of this system have been documented
10 //
11 // Revisions 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
12 //
13 //
14 //
15 //
16 //
17 //
18 //
19 //
20 //
21 // Included files --------
22 //
23 #include <iostream.h>
24 #include "stdtype.h"
25 #include "compoundstring.h"
26 //
27 // External Functions ---------
28 extern Word utility_not_capitalized(Word in_word);
29 //
30 // Calling Syntax --------------
31 //
32 Pattern PC_noun_cap_second_word(Phrase & in_phrase)
33 {
34     // Automatics - objects ---------
35     Pattern wildcard_pattern = in_phrase;
36     Word new_word;
37     // Automatics - other -------------
38     // Code begins ------------------
39     #ifdef DEBUG
40         cerr << "\t\t\tENTRY PC_noun_cap_second_word()" << endl;
41     #endif
42     // Get the current word from the second
43     // position of the input phrase, and convert
44     // it to lower case.
45     //
46     new_word = utility_not_capitalized( in_phrase.get_word(2) );
47     //
48     // Set the new word in the second position of
49     // the pattern, and return to the caller.
50     //
51     wildcard_pattern.set_word( 2, new_word );
52     return( WildCard_pattern );
53     //
54     //
55     //
56     //
57     //
58     //
59     //
60     //
61     //
62 }
#include <iostream.h>  
#include "stdtype.h"  
#include "compoundstring.h"  

extern pattern_fragment any_word;  
PATTERN PC_wrong_first_and_second_word(Phrase & in_phrase)  
{  
   //--------------- Automatics - objects ---------------/  
   Pattern wildcard_pattern = in_phrase;  
   Word new_word1;  
   Word new_word2;  
   //--------------- Automatics - other ------------------/  
   //--------------- Code begins ------------------------/  
   ifndef DEBUG  
      cerr << "\t\t\t\tENTRY PC_wrong_first_and_second_word()" << endl;  
   endif  
   //---------------/  
   // Create two pattern fragments which will/  
   // match any word, and insert them in the first/  
   // two word positions of the generated pattern.  
   new_word1.set_string(any_word);  
   new_word2.set_string(any_word);  
   wildcard_pattern.set_word(1, new_word1);  
   wildcard_pattern.set_word(2, new_word2);  
   return( wildcard_pattern );  
}
May 9 1993  construction/src/PC_wrong_first_word.C  Page 1

---

// File: PC_wrong_first_word.C
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file implements the pattern constructor function of
// generating a Pattern which will match the input Phrase,
// regardless of whether or not the first word in the
// Phrase actually matches.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
// Revisions By Reason
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

//--- Included files ---
#include <iostream.h>
#include "stdlib.h"
#include "compoundstring.h"

//--- External Data ---
extern pattern_fragment any_word;

//--- Calling Syntax ---
Pattern PC_wrong_first_word(Phrase & in_phrase)
{
  Pattern wildcard_pattern = in_phrase;
  Word new_word;

  //--- Automatics - objects ---
  #ifdef DEBUG
  cerr << "\t\tENTRY PC_wrong_first_word()" << endl;
  #endif

  // Generate a pattern fragment which will
  // match any word, and insert it in the first
  // position of the generated pattern.
  new_word.set_string(any_word);
  wildcard_pattern.set_word(1, new_word);
  return wildcard_pattern;
}
/**
 * @file    PC_wrong_fourth_word.C
 * @author  A. G. Mooney
 * @project German Language Tutor, RIT Master's Thesis
 * @date    May 9, 1993
 * @brief   This file implements the Pattern constructor function of
 *          generating a Pattern which will match the input Phrase,
 *          regardless of whether or not the fourth word in the
 *          Phrase actually matches.
 * @notes   The theory and design of this system have been documented
 *          in the accompanying thesis report.
 */

// --- Included files -----------------------------
#include <iostream.h>
#include "stdtype.h"
#include "compoundstring.h"

// --- External Data -----------------------------
extern pattern_fragment any_word;

// --- Calling Syntax -----------------------------
Pattern PC_wrong_fourth_word(Phrase & in_phrase)
{
    // --- Automatics - objects ------------------
    Pattern wildcard_pattern = in_phrase;
    Word new_word;

    // --- Automatics - other -------------------
    // Code begins -----------------------------
    
    #ifdef DEBUG
        cerr << "\t\t\tENTRY PC_wrong_fourth_word()" << endl;
    #endif

    // Create a pattern fragment which will match
    // any word, and insert it in the fourth word
    // position of the generated pattern.

    new_word.set_string(any_word);
    wildcard_pattern.set_word(4, new_word);
    return( wildcard_pattern );
}
May 9 1996

#include <iostream.h>
#include "shorttype.h"
#include "compoundstring.h"

extern pattern_fragment any_word;

Pattern PC_wrong_second_word(Phrase & in_phrase)
{
    Pattern wildcard_pattern = in_phrase;
    Word   new_word;

    /* Create a pattern fragment which will match 
     * any word, and insert it in the second word 
     * position of the generated pattern. */
    new_word.set_string(any_word);
    wildcard_pattern.set_word( 2, new_word );
    return( wildcard_pattern );
}
// File: PC_wrong_third_word.C
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file implements the pattern constructor function of
generating a Pattern which will Match the input phrase,
regardless of whether or not the third word in the
phrase actually matches.
// Notes: The theory and design of this system have been documented
in the accompanying thesis report.
// Revisions By Reason
// 26-Apr-93 Ken Staffan vi.0 release with final thesis report.

#include <iostream.h>
#include "stdtype.h"
#include "compoundstring.h"

extern pattern_fragment any_word;

Pattern PC_wrong_third_word(Phrase & in_phrase) {
    // Automatics - objects
    Pattern wildcard_pattern = in_phrase;
    Word new_word;
    // Automatics - other
    // Code begins

    ifdef DEBUG
cerr << "ENTRY PC_wrong_third_word()" << endl;
endif

    // Create a pattern fragment which will match
    // any word, and insert it into the third word
    // position of the generated pattern.
    new_word.set_string(any_word);
    wildcard_pattern.set_word(3, new_word);
    return wildcard_pattern;
}
May 9 1993 19:12:32 models/src/basephrase.C Page 1

1 //
2 // Filename: basephrase.C
3 // Project: German Language Tutor, RIT Master's Thesis
4 // Purpose: This file implements the BasePhrase class.
5 // Notes: The theory and design of this system have been documented
6 // in the accompanying thesis report.
7 //
8 // Revisions By Reason
9 // 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
10 //
11 //-------- Included files ---------------------------/
12 #include <iostream.h>
13 #include "basephrase.h"
14 #include "dictionary.h"
15 //-------- External data ---------------------------/
16 extern Dictionary dictionary;
17 // Constructor 1.
18 //-------- ---------------------------/
19 BasePhrase::BasePhrase(int in_length, int index1, int index2, int index3,
20 int index4, int index5, int index6, int index7,
21 int index8, int index9, int index10)
22 {#ifdef DEBUG
23 cerr << "\t\t\t\ENTRY BasePhrase CONSTRUCTOR 1" << endl;
24 cerr << "\t\t\t\ENTRY (for creating)" << endl;
25 #endif
26
27 int current_level;
28 int indices = new int[in_length];
29 int length = in_length;
30 knowledge_level = 0;
31 // For each word index which was input, store
32 // in the appropriate position in the
33 // internal array of indices, and check to see
34 // if it represents the highest knowledge
35 // level yet encountered. If so, adjust the
36 // current level, so that upon completion, the
37 // knowledge level of the BasePhrase is equal
38 // to the highest knowledge level of any of its
39 // words.
40 //-------- ---------------------------/
41 switch(length)
42 { case 0:
43     return;
44 case 1:
45     indices[0] = index1;
46     current_level = dictionary.get_level(index1);
47     if (current_level > knowledge_level)
48     { knowledge_level = current_level;
49     } case 2:
50     indices[1] = index2;
51     current_level = dictionary.get_level(index2);
52     if (current_level > knowledge_level)
53     { knowledge_level = current_level;
54     } case 3:
55     indices[2] = index3;
56     current_level = dictionary.get_level(index3);
57     if (current_level > knowledge_level)
58     { knowledge_level = current_level;
59     } case 4:
60     indices[3] = index4;
61     current_level = dictionary.get_level(index4);
62     if (current_level > knowledge_level)
63     { knowledge_level = current_level;
64     } case 5:
65     indices[4] = index5;
66     current_level = dictionary.get_level(index5);
67     if (current_level > knowledge_level)
68     { knowledge_level = current_level;
69     } case 6:
70     indices[5] = index6;
71     current_level = dictionary.get_level(index6);
72     if (current_level > knowledge_level)
73     { knowledge_level = current_level;
74     } case 7:
75     indices[6] = index7;
76     current_level = dictionary.get_level(index7);
77     if (current_level > knowledge_level)
78     { knowledge_level = current_level;
79     } case 8:
80     indices[7] = index8;
81     current_level = dictionary.get_level(index8);
82     if (current_level > knowledge_level)
83     { knowledge_level = current_level;
84     } case 9:
85     indices[8] = index9;
86     current_level = dictionary.get_level(index9);
87     if (current_level > knowledge_level)
88     { knowledge_level = current_level;
89     } case 10:
90     indices[9] = index10;
91     current_level = dictionary.get_level(index10);
92     if (current_level > knowledge_level)
93     { knowledge_level = current_level;
94     } case 11:
95     break;
96 default:
97     break;
98     return;
99     
100 //-------- ---------------------------/
101 BasePhrase::~BasePhrase()
102 {#ifdef DEBUG
103     cerr << "\t\t\t\ENTRY BasePhrase DESTRUCTOR 1" << endl;
104 #endif
105     // Free internal memory used by BasePhrase
106     delete [] indices;
107     // Return the knowledge level to
108     // the dictionary
109     dictionary.set_level(knowledge_level);
110 
111 //-------- ---------------------------/
112 BasePhrase::~BasePhrase(int in_length, int index1, int index2, int index3,
113 int index4, int index5, int index6, int index7,
114 int index8, int index9, int index10)
115 {#ifdef DEBUG
116     cerr << "\t\t\t\ENTRY BasePhrase CONSTRUCTOR 1" << endl;
117     cerr << "\t\t\t\ENTRY (for destroying)" << endl;
118 #endif
119
120     // Free internal memory used by BasePhrase
121     delete [] indices;
122     // Return the knowledge level to
123     // the dictionary
124     dictionary.set_level(knowledge_level);
125     //-------- ---------------------------/
126     return;
127 //-------- ---------------------------/
128 BasePhrase& BasePhrase::operator=(BasePhrase& other)
129 {#ifdef DEBUG
130     cerr << "\t\t\t\ENTRY BasePhrase ASSIGNMENT" << endl;
131 #endif
132     // Free internal memory used by BasePhrase
133     delete [] indices;
134     // Copy the knowledge level of the other BasePhrase
135 basephrase.C Page 2
136     }
```c
133  
134  
135  #ifdef DEBUG
136  
137  cout << "\t\tERROR BasePhrase invalid length specified" << endl;
138  
139  #endif
140  
141  if ((in_BP != NULL) && (length >= 0))
142  
143  if ((in_BP != NULL) && (length >= 0))
144  
145  BasePhrase::BasePhrase(BasePhrase & in_BP)
146  
147  #ifdef DEBUG
148  
149  
150  
151  if ((in_BP != NULL) && (length >= 0))
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  BasePhrase& BasePhrase::operator=(BasePhrase & in_BP)
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
```
return(length);

// Member function get_level().

int BasePhrase::get_level()
{
   #ifdef DEBUG
      cerr << " t t t t ENTRY BasePhrase::get_level" << endl;
   #endif

   //-------- Automatics - objects ----------/
   //-------- Automatics - other ----------/
   //-------- Code begins ---------------/

   return(knowledge_level);
}
// Member function get_BasePhrase().

// BasePhraseList::get_BasePhrase()

#define DEBUG

cerr << "\t\t\tENTRY BasePhraseList::get_BasePhrase" << endl;
#endif

// Automatics - objects

// Automatics - other

BasePhrase * BP_ptr = NULL;
int num_unselected = 0;
int rand_val = rand();
int chosen_count;
int chosen_index;

// Code begins

// First, determine how many BasePhrases in the
// list have not yet been selected.

for (int i=0; i<size; i++)
{
    if (selected[i] == FALSE)
    {
        num_unselected++;
    }
}

// If there are no unused BasePhrases, do
// nothing (BP_ptr of NULL will be returned to
// the caller). Otherwise, pick a pseudo-
// random number in the range of available
// BasePhrases.

if (num_unselected != 0)
{
    chosen_count = rand_val % num_unselected;
    #ifdef DEBUG
    cerr << "\t\t\tDEBUG number of unselected entries is " << num_unselected << endl;
    cerr << "\t\t\tDEBUG random value obtained is " << rand_val << endl;
    cerr << "\t\t\tDEBUG chosen count is modulo above 2 and is " << chosen_count << endl;
    #endif
    chosen_count++;
    #ifdef DEBUG
    cerr << "\t\t\tDEBUG chosen count final value is " << chosen_count << endl;
    #endif
    // Use this chosen random index to count into
    // previously unselected entries.

    for (chosen_index=0; chosen_count > 0; chosen_index++)
    {
        // Destructor.
        BasePhraseList::~BasePhraseList();
        #ifdef DEBUG
        cerr << "\t\t\tENTRY BasePhraseList DESTRUCTOR" << endl;
        #endif
        delete [] phrases;
        delete [] selected;
    }
if (selected[chosen_index] == FALSE) {
    chosen_count--; 
}

if (chosen_index != NULL) {
    chosen_index--; 
}

//NOTE: get_index currently does some debug output, so let it.
int temp;

temp = phrases[chosen_index] -> get_index(1);

temp = phrases[chosen_index] -> get_index(2);

if (number_unselected > 0) {
    return(TRUE);
}

//selected, mark it as "used", and return the
//BasePhrase pointer to the caller.

// Now that a pseudo-random entry has been
// selected, mark it as "used", and return the
// BasePhrase pointer to the caller.

selected[chosen_index] = TRUE;

BP_ptr = phrases[chosen_index];

return(BP_ptr);
May 9 1993 19:12:3 models/src/basephrasepool.C  Page 1

1  //---------------------------------------------
2  //  Filename:    basephrasepool.C
3  //  Project:    German Language Tutor, RIT Master's Thesis
4  //  Purpose:    This file implements the BasePhrasePool class.
5  //  Notes:      The theory and design of this system have been documented
6  //                in the accompanying thesis report.
7  //                It would probably be nice if the constructor took a pointer
8  //                to a variable length list of BasePhraseLists, as opposed
9  //                to the individual arguments currently used.
10 //
11 //  Revisions By   Reason
12 //   --------   -------
14 //  --------   -------
15 //
16 //-------- Included files ------------------/
17 #include <iostream.h>
18 #include "basephrasepool.h"
19 #include <stdlib.h>
20 // Constructor.
21 //
22 BasePhrasePool::BasePhrasePool(int pool_size, BasePhraseList * list1,
23    BasePhraseList * list2, BasePhraseList * list3,
24    BasePhraseList * list4, BasePhraseList * list5,
25    BasePhraseList * list6, BasePhraseList * list7,
26    BasePhraseList * list8, BasePhraseList * list9,
27    BasePhraseList * list10)
28 {  
29     #ifdef DEBUG
30     cerr << "\t\t\tENTRY BasePhrasePool CONSTRUCTOR" << endl;
31     #endif
32     size = pool_size;
33     list = new BasePhraseList * [size];
34     selected = new bool[size];
35     //------------------------------------------------------------------------------
36     // Mark all BasePhraseList choices as initially //
37     // unselected (meaning not yet exhausted).
38     //------------------------------------------------------------------------------
39     for (int i = 0; i < pool_size; i++)
40     {  
41         selected[i] = FALSE;
42     }
43     //------------------------------------------------------------------------------
44     // Set list entries based on input.
45     //------------------------------------------------------------------------------
46     switch(size)
47     {  
48         case 10:  
49             lists[9] = list10;
50         case 9:  
51             lists[8] = list9;
52         case 8:  
53             lists[7] = list8;
54         case 7:  
55             lists[6] = list7;
56         case 6:  
57             lists[5] = list6;
58         case 5:  
59             lists[4] = list5;
60         case 4:  
61             lists[3] = list4;
62         case 3:  
63             lists[2] = list3;
64         case 2:  
65             lists[1] = list2;
66         case 1:  
67             lists[0] = list1;
68         break;
69     }
70     #ifdef DEBUG
71     cout << "\t\t\tENTRY BasePhrasePool DESTRUCTOR" << endl;
72     #endif
73     delete [] lists;
74     delete [] selected;
75     }  
76  #ifdef DEBUG
77  cerr << "\t\t\tENTRY BasePhrasePool::get_BasePhrase()" << endl;
78  #endif
79  //------------------------------------------------------------------------------
80  BasePhrase * BasePhrasePool::get_BasePhrase()
81  {  
82     //------------------------------------------------------------------------------
83     #ifdef DEBUG
84     cerr << "\t\t\tENTRY BasePhrasePool::get_BasePhrase" << endl;
85     #endif
86     //------------------------------------------------------------------------------
87     //-------- Automatics = objects --------------------------
88     //------------------------------------------------------------------------------
89     //-------- Automatics = other ---------------------------
90     //------------------------------------------------------------------------------
91     BasePhrase * BP_ptr = NULL;
92     bool still_looking = TRUE;
93     int num_unselected = 0;
94     int rand_val;
95     int chosen_count;
96     int chosen_index;
97     //------------------------------------------------------------------------------
98  }
for (int i=0; i<size; i++)
    if (selected[i] == FALSE)
        num_unselected++;

while (still_looking)
    if (num_unselected == 0)
        still_looking = FALSE;
    else
        rand_val = rand();
        chosen_count = rand_val % num_unselected;
        if (DEBUG)
            cerr << "DEBUG number of unselected entries is " << num_unselected << endl;
        cerr << "DEBUG random value obtained is " << rand_val << endl;
        cerr << "DEBUG chosen count is modulo above 2 and is " << chosen_count << endl;
    
    if (DEBUG)
        cerr << "DEBUG chosen count final value is " << chosen_count << endl;
    
    // Use the chosen random index to count into
    // the lists to request a BasePhrase.
    //
    for (chosen_index=0; chosen_count > 0; chosen_count++)
        if (selected[chosen_index] == FALSE)
            chosen_count--;
    chosen_index--;

    if (DEBUG)
        cerr << "DEBUG using the list at array location " << chosen_index << endl;
    
    // A BasePhraseList may report that it has
    // no more unused BasePhrases at this point.
    // If this happens, mark the Lists entry as

    // exhausted, and continue looking.
    //
    // new unused
    if (BP_ptr == NULL)
        num_unselected--;
    selected[chosen_index] = TRUE;
    still_looking = TRUE;
    else
        still_looking = FALSE;

    if (DEBUG)
        cerr << "DEBUG in BP::reset_sel, resetting sel for list " << i << endl;
    
    if (DEBUG)
        cerr << "DEBUG BPL reports it has active BPs. " << endl;
    
    if (DEBUG)
        cerr << "DEBUG BPL reports it does not have active BPs. " << endl;

    // Member function reset_selections().
    //
    // How to get a pseudo-random number in the range
    // of currently non-exhausted lists.
    //
    rand_val = rand();
    chosen_count = rand_val % num_unselected;
    if (DEBUG)
        cerr << "DEBUG number of unselected entries is " << num_unselected << endl;
    cerr << "DEBUG random value obtained is " << rand_val << endl;
    cerr << "DEBUG chosen count is modulo above 2 and is " << chosen_count << endl;
    
    if (DEBUG)
        cerr << "DEBUG chosen count final value is " << chosen_count << endl;
    
    // Use the chosen random index to count into
    // the lists to request a BasePhrase.
    //
    for (chosen_index=0; chosen_count > 0; chosen_count++)
        if (selected[chosen_index] == FALSE)
            chosen_count--;
    chosen_index--;

    if (DEBUG)
        cerr << "DEBUG using the list at array location " << chosen_index << endl;
    cerr << "DEBUG chosen index is " << chosen_index << endl;

    // A BasePhraseList may report that it has
    // no more unused BasePhrases at this point.
    // If this happens, mark the Lists entry as

    // exhausted, and continue looking.
    //
259                      //-------------------------------/
260                      // If the reset process causes all BasePhrase  //
261                      // Lists to report that they have no available //
262                      // BasePhrase selections, return FALSE to the  //
263                      // caller.                                  //
264                      //-------------------------------/
265                      //
266    if (number_active_lists > 0)  
267    {                             
268   return(TRUE);                 
269    }                            
270    else                          
271    {                            
272      return(FALSE);             
273    }                            
274    }
May 9 1993 19:10:00

---

1 # File: CompoundString.C
2 # Project: German Language Tutor, RIT Master's Thesis
3 # Purpose: This file implements the CompoundString class, and its
4 # derived classes Phrase and Pattern.
5 # Notes: The theory and design of this system have been documented
6 # in the accompanying thesis report.
7 # Revisions By Reason
8 # 26-Apr-93 Ken Staffan v1.0 release with final thesis report.
9
10 /// Regular expression processing defns -------
11 #define INIT register char *sp = in_str;
12 #define GETC() (*sp++);
13 #define PEEKC() (*sp)
14 #define UNGETC(c) (--sp)
15 #define RETURN(c) return(c);
16 #define ERROR(c) printf("regexp\ error to\n",c)
17 #define EXPBUF_SIZE 250
18
19 //-------- Included files -------------------
20 #include <iostream.h>
21 #include "compoundstring.h"
22 #include "word.h"
23 #include <stdio.h>
24 #include <regex.h>
25 #include <string.h>
26 #include "word_and_phrase_defs.h"
27
28 //--- Base class -----------------------------
29
30 CompoundString::CompoundString()
31 {
32 #ifdef DEBUG
33    cerr << \t\t\t\tENTRY CompoundString CONSTRUCTOR* << endl;
34 #endif
35
36 // Create an array of word pointers, and
37 // initialize all entries to NULL
38 size = MAX_PHRASE_LENGTH;
39 words = new Word *[size];
40 for (int i=0; i<size; i++)
41 {
42     words[i] = NULL;
43 }
44 #ifdef DEBUG
45    // Destructor.
46 #endif
47
48 CompoundString::CompoundString()
49 {
50 #ifdef DEBUG
51    cerr << \t\t\t\tENTRY CompoundString DESTRUCTOR* << endl;
52 #endif
53
54 CompoundString& CompoundString::operator=(CompoundString & in_CS)
55 {
56 #ifdef DEBUG
57    cerr << \t\t\t\tENTRY CompoundString:operator* = << endl;
58 #endif
59
60 // If assignment to itself, don't do anything, //
61 // otherwise, loop through the word array, //
62 // deleting any words found. Then delete //
63 // the array itself.
64 #ifdef DEBUG
65    #if (in_CS != this)
66        for (int i=0; i<size; i++)
67           if (words[i] != NULL)
68             [delete words[i];
69         }
70 #endif
71 }
72
73 // Create a new words array, of size to match //
74 // the CompoundString on the right-hand side //
75 // of the assignment. Loop through the right- //
76 // hand Compound string, and create new word //
77 // entries to match any found.
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132


```c
size = in.size;
words = new Word * [size];
for (int i=0; i<size; i++)
|

words[i] = NULL;
}

size = in_phrase.size;
words = new Word * [size];
for (int i=0; i<size; i++)
|

if (in_phrase.words[i] == NULL)
words[i] = NULL;
else
words[i] = new Word(*(in_phrase.words[i]));

size = in_phrase.size;
words = new Word * [size];
for (int j=0; j<size; j++)
|

if (in_phrase.words[j] == NULL)
words[j] = new Word(*(in_phrase.words[j]));
```


```c
if (in_phrase != this)
for (int i=0; i<size; i++)
|

if (words[i] == NULL)
delete words[i];
}
delete [] words;

size = in_phrase.size;
words = new Word * [size];
for (int j=0; j<size; j++)
|

if (in_phrase.words[j] == NULL)
words[j] = new Word(*(in_phrase.words[j]));
```

```c
    return *this;
    }
    return this;
```

```c
    // Destructor.
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
    
    // --phrase
```
May 9 1993 19:37:43

    boolean end_of_line = FALSE;
    
    //--- Code begins ----------------------/
    //--- Since the entire Phrase will be read from
    //--- the user's input - delete any Words which
    //--- are already assigned.
    //---
    for (int i=0; i<size; i++)
        {  
            if (words[i] != NULL)
                { 
                    delete words[i];
                }
            }
    // Eat any leading white space.
    //---
    char ch;
    boolean end_of_whitespace = FALSE;
    while (!end_of_whitespace & cin.get(ch))
        {  
            switch(ch)
                {  
                    case '\t':
                    case '\n':
                        break;
                    default:
                        cin.putback(ch);
                        end_of_whitespace = TRUE;
                        break;
                }
    //--- Allow individual Words to be read until the
    //--- end of the line is encountered.
    //---
    for (int j=0; end_of_line = FALSE, j++)
        { 
            end_of_line = new_words[j]->read();
            words[j] = new Word(new_words[j]);
        }
    size = j;
    //---
    // Member function print().
    //---
    void Phrase::print()
    {  
        if (ok)  
            {  
                for (int i=0; i<size; i++)
                    {  
                        if (1 != 0)
                        {  
                            cout << " ";
                        }
                        (*words[i]).print();
                    }
            }
    else
        {  
            cerr << "ERROR trying to print incomplete phrase" << endl;
            #ifdef DEBUG
            cerr << "\t\t\tENTRY Phrase::print" « endl;
            #endif
        }
    //--- Automatics - objects ----------/

---
May 9 1993 19:00 amused/src/compoundstring.C Page 9

```c
328 {
329     //--------- Automatics - objects -------------
330     //--------- Automatics - other -------------
331     boolean ok = TRUE;
332     //------ Code begins -----------------------
333     // This function is essentially the same as
334     // print(), but it uses Word::debug_print,
335     // which includes diagnostic output.
336     //
337     for (int i=0; i<size; i++)
338     {
339         if (words[i] == NULL)
340             { ok = FALSE; break; }
341         }
342     }
343     if (ok)
344     {
345         for (int i=0; i<size; i++)
346             {
347                 if (i != 0)
348                     { cerr << ' ';
349                     }
350                 cerr << *(words[i]).debug_print();
351             }
352         }
353     //------ Member function get_size() -------
354     //------ Constructor 2 of 2 ---------------
355     int Phrase::get_size()
356     { #ifdef DEBUG
357         cerr << "\t\t\t\tENTRY Phrase::get_size" << endl;
358         #endif
359         return(size);
360     }
361     //------ Derived class -------------------
362     // Constructor 1 of 2. 
363     //------ Pattern::Pattern(const Phrase & in_phrase) 
364     { #ifdef DEBUG
365         cerr << "\t\t\t\tENTRY Pattern::Pattern" << endl;
366         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR 1" << endl;
367         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR 2" << endl;
368         cerr << "\t\t\t\tENTRY (for copies)" << endl;
369         #endif
370         size = in_pattern.size;
371         words = new Word * [size];
372         for (int i=0; i<size; i++)
373             { if (in_pattern.words[i] == NULL)
374                 words[i] = NULL;
375             else
376                 words[i] = new Word(*(in_pattern.words[i]));
377             }
378     #ifdef DEBUG
379         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR done..." << endl;
380         #endif
381     }
382     //------ Pattern::Pattern(const Pattern & in_pattern) 
383     { #ifdef DEBUG
384         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR 2" << endl;
385         cerr << "\t\t\t\tENTRY (for copies)" << endl;
386         #endif
387         //------ Pattern::Pattern(const Phrase & in_phrase) 
388     { #ifdef DEBUG
389         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR 1" << endl;
390         cerr << "\t\t\t\tENTRY (for autos from Phrase)" << endl;
391         #endif
392     #ifdef DEBUG
393     #endif DEBUG
```

May 9 1993 19:00 amused/src/compoundstring.C Page 10

```c
595     cerr << "\t\t\t\tDEBUG Phrase to construct pattern from is: " << endl;
596     in_phrase.debug_print();
597     #endif
598     #ifdef DEBUG
599     cerr << "\t\t\t\tDEBUG Phrase to construct pattern from is: " << endl;
600     #endif
601     #ifdef DEBUG
602     cerr << "\t\t\t\tENTRY Phrase:in_phrase.print" << endl;
603     #endif
604     //------ Code begins -----------------------
605     //------ Code begins -----------------------
606     #ifdef DEBUG
607     cerr << "\t\t\t\tENTRY Phrase:in_phrase.size" << endl;
608     #endif
609     size = in_phrase.size;
610     #ifdef DEBUG
611     cerr << "\t\t\t\tENTRY Phrase:in_phrase.size is full" << endl;
612     #endif
613     #ifdef DEBUG
614     cerr << "\t\t\t\tENTRY Phrase:in_phrase.size will be *" << endl;
615     #endif
616     words = new Word * [size];
617     for (int i=0; i<size; i++)
618         { if (in_phrase.words[i] == NULL)
619             words[i] = NULL;
620         }
621     else
622         { words[i] = new Word(*(in_phrase.words[i]));
623         }
624     #ifdef DEBUG
625     cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR done..." << endl;
626     cerr << "\t\t\t\tENTRY Pattern constructor done..." << endl;
627     #endif
628     #ifdef DEBUG
629     cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR done..." << endl;
630     #endif
631     //------ Derived class -------------------
632     //------ Derived class -------------------
633     //------ Pattern::Pattern(const Pattern & in_pattern) 
634     { #ifdef DEBUG
635         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR 2" << endl;
636         cerr << "\t\t\t\tENTRY (for copies)" << endl;
637         #endif
638         //------ Pattern::Pattern(const Phrase & in_phrase) 
639     { #ifdef DEBUG
640         cerr << "\t\t\t\tENTRY Pattern CONSTRUCTOR 1" << endl;
641         #endif
642     #ifdef DEBUG
643     #endif DEBUG
```
```c
if (in_pattern.words[i] != NULL)
```
// Use the regexp() library capabilities to
// compile the pattern and attempt the match.

compile(pattern, expbuf, expbuf+EXPBUF_SIZE, eof);

if (step(phrase, expbuf))
    matched = TRUE;
#endif DEBUG
    cerr << "\t\tDEBUG match returning TRUE" << endl;
#else DEBUG
    cerr << "\t\tDEBUG match returning FALSE" << endl;
#endif DEBUG

return(matched);
#include "stdtype.h"

pattern_fragment any_word = "\<[a-zA-Z]\>|[a-zA-Z]*\"";
DictionaryEntry_adjective adj_list[] =
{
    DictionaryEntry_adjective(200001,1,"good","gut"),
    DictionaryEntry_adjective(200002,1,"warm","warm"),
    DictionaryEntry_adjective(200003,1,"young","jung"),
    DictionaryEntry_adjective(200004,1,"round","rund"),
    DictionaryEntry_adjective(200005,1,"cold","kalt"),
    DictionaryEntry_adjective(200006,1,"old","alt"),
    DictionaryEntry_adjective(200007,1,"new","neu"),
    DictionaryEntry_adjective(200008,1,"blue","blau"),
    DictionaryEntry_adjective(200009,1,"brown","braun"),
    DictionaryEntry_adjective(200010,1,"long","lang"),
    DictionaryEntry_adjective(200011,1,"here","hier"),
    DictionaryEntry_adjective(200012,1,"full","voll"),
    DictionaryEntry_adjective(200013,1,"clear","klar"),
    DictionaryEntry_adjective(200014,1,"fresh","frisch"),
    DictionaryEntry_adjective(200015,1,"red","rot"),
    DictionaryEntry_adjective(200016,1,"tall","groß"),
    DictionaryEntry_adjective(200017,1,"there","dort")
};

// Verbs. For reference, the initial arguments //
// are the word index, the knowledge level, //
// and the English and German present tense. //

DictionaryEntry_verb verb_list[] =
{
    DictionaryEntry_verb(300001,2,"is","ist"),
    DictionaryEntry_verb(300002,1,"sings","singt"),
    DictionaryEntry_verb(300003,1,"sends","sendet"),
    DictionaryEntry_verb(300004,1,"finds","findet"),
    DictionaryEntry_verb(300005,1,"fauls","faellt"),
    DictionaryEntry_verb(300006,1,"springs","springt"),
    DictionaryEntry_verb(300007,1,"binds","bindet"),
    DictionaryEntry_verb(300008,1,"brings","bringt"),
    DictionaryEntry_verb(300009,1,"begins","beginnt"),
    DictionaryEntry_verb(300010,1,"sees","sieht"),
    DictionaryEntry_verb(300011,1,"washes","wascht"),
    DictionaryEntry_verb(300012,1,"helps","hilft"),
    DictionaryEntry_verb(300013,1,"has","hat"),
    DictionaryEntry_verb(300014,1,"comes","kommt"),
    DictionaryEntry_verb(300015,1,"drinks","trinkt"),
    DictionaryEntry_verb(300016,1,"goes","geht"),
    DictionaryEntry_verb(300017,2,"learns","lernt"),
    DictionaryEntry_verb(300018,2,"plays","spielt"),
    DictionaryEntry_verb(300019,2,"stays","steht"),
    DictionaryEntry_verb(300020,2,"rolls","rollt")
};

Dictionary dictionary(noun_list, article_list, adj_list, verb_list);
May 9 1993 19: models/src/data_phrasemodellist.C  Page 1

1  
2  //
3  //  Filename:  data_phrasemodellist.C
4  //
5  //  Project:  German Language Tutor, RIT Master's Thesis
6  //
7  //  Purpose:  This file contains no executable code.  Here, all the phrase
8  //  models for the phrase model list are assembled (part of the
9  //  system's knowledge base).
10  //
11  //  Notes:  The theory and design of this system have been documented
12  //  in the accompanying thesis report.
13  //
14  //  Even with the current limited comments, this file is
15  //  quickly becoming unreadable [e.g. see the PatternNode
16  //  definitions below].  It would be desirable to devise a
17  //  more readable format, or even better, to provide tools
18  //  which allow for viewing and modifying the data structures
19  //  in a friendlier way.
20  //
21  //  Revisions By  Reason
22  //  ------  -------
24  //
25  //
26  //  ----------  Included files  ----------
27  #include <iostream.h>
28  #include "dictionary.h"
29  #include "typedef.h"
30  #include "basephrase.h"
31  #include "basephaselist.h"
32  #include "basephrasepool.h"
33  #include "phasemodel.h"
34  #include "worddescriptor.h"
35  #include "phasemodellist.h"
36  
37  extern Dictionaty dictionary;
38  
39  //----------  External functions  ----------
40  
41  extern Pattern PC_gender1_first_word(Phrase 4);
42  extern Pattern PC_gender2_first_word(Phrase 4);
43  extern Pattern PC_nouncap_second_word(Phrase 4);
44  extern Pattern PC_wrong_first_and_second_word(Phrase 4);
45  extern Pattern PC_wrong_first_word(Phrase 4);
46  extern Pattern PC_wrong_fourth_word(Phrase 4);
47  extern Pattern PC_wrong_second_word(Phrase 4);
48  extern Pattern PC_wrong_third_word(Phrase 4);
49  extern Pattern PC_extra_article_fourth(Phrase 4);
50  
51  extern BasePhrase BPF_no_translation(BasePhrase 4);
52  extern BasePhrase BPF_drop_fourth_word(BasePhrase 4);
53  
54  //----------  The Base Phrases  ----------
55  

67  #define BPF_0001_SIZE 4
68  #define BPF_0001_SIZE 12
69  BasePhrase Bp_0001() =
70  
71  { // the man is good
72  BasePhrase(BP_0001_SIZE, 100001, 1, 300001, 200001),
73  // the bread is warm
74  BasePhrase(BP_0001_SIZE, 100001, 7, 300001, 200002),
75  // the beer is cold
76  BasePhrase(BP_0001_SIZE, 100001, 37, 300001, 200005),
77  // the house is old
78  BasePhrase(BP_0001_SIZE, 100001, 13, 300001, 200006),
79  // the plan is new
80  BasePhrase(BP_0001_SIZE, 100001, 27, 300001, 200007),
81  // the below
82  BasePhrase(BP_0001_SIZE, 100001, 34, 300001, 200008),
83  // the hat is brown
84  BasePhrase(BP_0001_SIZE, 100001, 12, 300001, 200009),
85  // the grass is long
86  BasePhrase(BP_0001_SIZE, 100001, 8, 300001, 200010),
87  // the glass is full
88  BasePhrase(BP_0001_SIZE, 100001, 28, 300001, 200012),
89  // the sea is clear
90  BasePhrase(BP_0001_SIZE, 100001, 31, 300001, 200013),
91  // the chair is red
92  BasePhrase(BP_0001_SIZE, 100001, 43, 300001, 200015),
93  // the woman is tall
94  BasePhrase(BP_0001_SIZE, 100001, 25, 300001, 200016)
95  
96  }
97  
98  #define BPF_0002_SIZE 4
99  #define BPF_0002_SIZE 2
100 BasePhrase Bp_0002() =
101  
102  { // the mother is young
103  BasePhrase(BP_0002_SIZE, 100001, 6, 300001, 200003),
104  // the table is round
105  BasePhrase(BP_0002_SIZE, 100001, 5, 300001, 200004)
106  
107  }
108  
109  #define BPF_0003_SIZE 4
110  #define BPF_0003_SIZE 4
111  BasePhrase Bp_0003() =
112  
113  { // the girl learns German
114  BasePhrase(BP_0003_SIZE, 100001, 4, 300001, 51),
115  // the doctor drinks coffee
116  BasePhrase(BP_0003_SIZE, 100001, 50, 300001, 47),
117  // the garden has roses
118  BasePhrase(BP_0003_SIZE, 100001, 16, 300001, 38),
119  // the teacher brings paper
120  BasePhrase(BP_0003_SIZE, 100001, 35, 300008, 24)
121  
122  }
123  
124  #define BPF_0004_SIZE 5
125  #define BPF_0004_SIZE 1
126  BasePhrase Bp_0004() =
127  
128  { // the man is a merchant
129  BasePhrase(BP_0004_SIZE, 100001, 1, 300001, 100002, 52)
130  
131  }
132  
133  
134  //----------  The Base Phrase Lists  ----------
135  
136

134 BasePhraseList bpl_0001( BPL_0001_SIZE, bp_0001 );
135 BasePhraseList bpl_0002( BPL_0002_SIZE, bp_0002 );
136 BasePhraseList bpl_0003( BPL_0003_SIZE, bp_0003 );
137 BasePhraseList bpl_0004( BPL_0004_SIZE, bp_0004 );
138
139 //-------- The Base Phrase Pools --------
140
141 typedef BPP_0001_SIZE 1
142 BasePhrasePool bpp_0001( BPP_0001_SIZE,
143                           bppl_0001 );
144
145 typedef BPP_0002_SIZE 2
146 BasePhrasePool bpp_0002( BPP_0002_SIZE,
147                           bppl_0002 );
148
149 typedef BPP_0003_SIZE 3
150 BasePhrasePool bpp_0003( BPP_0003_SIZE,
151                           bppl_0003 );
152
153 typedef BPP_0004_SIZE 4
154 BasePhrasePool bpp_0004( BPP_0004_SIZE,
155                           bppl_0004 );
156
157 //-------- The Word Descriptors --------
158
159 WordDescriptor wd_0001( C_singular, G_DYNAMIC, L_english, article);
160 WordDescriptor wd_0002( C_singular, G_masculine, L_english, noun );
161 WordDescriptor wd_0003( C_singular, G_masculine, L_german, article);  
162 WordDescriptor wd_0004( C_singular, G_masculine, L_german, noun );
163
164 WordDescriptor wd_0005( C_none, G_none, L_english, adjective);
165 WordDescriptor wd_0006( C_none, G_none, L_english, verb );
166 WordDescriptor wd_0007( C_none, G_none, L_german, adjective);
167 WordDescriptor wd_0008( C_none, G_none, L_german, verb );
168
169 //-------- The Phrase Descriptors --------
170
171 typedef PD_SIZE_0001 4
172 int pd_0001_order[] = { 1, 3, 4, 2 };  
173 PhraseDescriptor pd_0001( PD_SIZE_0001, pd_0001_order, wdp_0001, wdp_0002,
174                           wdp_0003, wdp_0004, wdp_0005 );
175
176 typedef PD_SIZE_0002 4
177 int pd_0002_order[] = { 2, 1, 3, 4 };  
178 PhraseDescriptor pd_0002( PD_SIZE_0002, pd_0002_order, wdp_0001, wdp_0002,
179                           wdp_0003, wdp_0004, wdp_0005 );
180
181 typedef PD_SIZE_0003 5
182 int pd_0003_order[] = { 2, 1, 3, 4, 5 };  
183 PhraseDescriptor pd_0003( PD_SIZE_0003, pd_0003_order, wdp_0001, wdp_0002,
184                           wdp_0003, wdp_0004, wdp_0005, wdp_0006 );
185
186 typedef PD_SIZE_0004 5
187 int pd_0004_order[] = { 2, 1, 3, 5, 4 };  
188 PhraseDescriptor pd_0004( PD_SIZE_0004, pd_0004_order, wdp_0001, wdp_0002,
189                           wdp_0003, wdp_0004, wdp_0005, wdp_0006, wdp_0007 );
190
191 //-------- The Translation Trees --------
192
193 TranslationTree tt_0001( BPT_no_translation, &pt_0001, &pd_0001 );
194 TranslationTree tt_0002( BPT_no_translation, &pt_0002, &pd_0002 );
195 TranslationTree tt_0003( BPT_drop_fourth_word, &pt_0011, &pd_0061 );
196
197 //-------- The Phrase Models --------
198
199 #define PML_0001_SIZE 4
200 PhraseModel pml_0001[] =  
201 { PhraseModel(7, bpp_0001, pd_0001, tct_0001),
202     PhraseModel(3, bpp_0002, pd_0002, tct_0002),
203     PhraseModel(5, bpp_0004, pd_0005, tct_0003) }
204
205 PhraseModelList models( PML_0001_SIZE, pml_0001 );

May 9 1993 19: models/src/data_phrasemodellist.C  Page 4

199 //-------- The Patterns --------
200
201 Hint hint_0001("The first word is not what was expected.");
202 Hint hint_0002("The second word is not what was expected.");
203 Hint hint_0003("The third word is not what was expected.");
204 Hint hint_0004("The fourth word is not what was expected.");
205 Hint hint_0005("The first and second words are not what were expected.");
206 Hint hint_0006("Nouns in German should be capitalized.");
207 Hint hint_0007("The gender of the article is not correct.");
208 Hint hint_0008("The gender of the article is not correct.");
209 Hint hint_0019("An article that should not be included in this case.");
210
211 PatternNode pat_0008( RCP_gender_first_word, NULL, NULL, &hint_0008);
212 PatternNode pat_0007( RCP_gender_first_word, NULL, &pat_0008, &hint_0007);
213 PatternNode pat_0006( RCP_noun_cap_second_word, NULL, NULL, &hint_0006);
214 PatternNode pat_0005( RCP_wrong_first_and_second_word, NULL, NULL, &hint_0005 );
215 PatternNode pat_0004( RCP_wrong_fourth_word, NULL, &pat_0005, &hint_0004);
216 PatternNode pat_0003( RCP_wrong_third_word, NULL, &pat_0004, &hint_0003);
217 PatternNode pat_0002( RCP_wrong_second_word, &pat_0006, &pat_0003, &hint_0002);
218 PatternNode pat_0001( RCP_wrong_first_word, &pat_0007, &pat_0002, &hint_0001);
219
220 PatternNode pat_0019( RCP_extra_article_fourth, NULL, NULL, &hint_0019);
221 PatternNode pat_0018( RCP_gender_first_word, NULL, &hint_0018, &hint_0008);
222 PatternNode pat_0017( RCP_gender_first_word, NULL, &pat_0018, &hint_0007);
223 PatternNode pat_0016( RCP_noun_cap_second_word, NULL, NULL, &hint_0006);
224 PatternNode pat_0015( RCP_wrong_first_and_second_word, NULL, &pat_0019, &hint_0005 );
225 PatternNode pat_0014( RCP_wrong_fourth_word, NULL, &pat_0015, &hint_0004);
226 PatternNode pat_0013( RCP_wrong_third_word, NULL, &pat_0014, &hint_0003);
227 PatternNode pat_0012( RCP_wrong_second_word, &pat_0016, &pat_0013, &hint_0002);
228 PatternNode pat_0011( RCP_wrong_first_word, &pat_0017, &pat_0012, &hint_0001);
229
230 Hint hint_0001("The gender of the first word is wrong.");
231 Hint hint_0002("There is something wrong with the second word in your response.");
232
233 PatternNode pat_0002( RCP_wrong_second_word, NULL, NULL, &hint_0002);
234 PatternNode pat_0003( RCP_wrong_first_word, &pat_0001, &pat_0002, &hint_0003);
235
236 //-------- The Translation Trees --------
237
238 TranslationTree tt_0001( BPT_no_translation, &pt_0001, &pd_0001 );
239 TranslationTree tt_0002( BPT_no_translation, &pt_0002, &pd_0002 );
240 TranslationTree tt_0003( BPT_drop_fourth_word, &pt_0011, &pd_0061 );
241
242 //-------- The Phrase Models --------
243
244 #define PML_0001_SIZE 4
245 PhraseModel pml_0001[] =  
246 { PhraseModel(7, bpp_0001, pd_0001, tct_0001),
247     PhraseModel(3, bpp_0002, pd_0002, tct_0002),
248     PhraseModel(5, bpp_0004, pd_0005, tct_0003) 
249     };
250
251 PhraseModelList models( PML_0001_SIZE, pml_0001 );
# Dictionary Class

**Dictionary::Dictionary**

```cpp
// Constructor.

Dictionary::Dictionary(DictionaryEntry_noun * noun_list,
                        DictionaryEntry_noun * adj_list,
                        DictionaryEntry_adj * verb_list)
{
    #ifdef DEBUG
    cerr << "\t\t\t\t\tENTRY Dictionary CONSTRUCTOR* << endl;
    #endif

    noun = noun_list;
    articles = article_list;
    adjs = adj_list;
    verbs = verb_list;
}
```

**Dictionary::~Dictionary**

```cpp
// Destructor.

Dictionary::~Dictionary()
{
    #ifdef DEBUG
    cerr << "\t\t\t\t\tENTRY Dictionary DESTRUCTOR* << endl;
    #endif

    // Member function get_word().

    Word Dictionary::get_word(int word_index, WordDescriptor * word_desc)
    {
        #ifdef DEBUG
        cerr << "\t\t\t\t\tENTRY Dictionary::get_word* << endl;
        #endif

        // Determine the indicated word type, in order
        // to search the appropriate DictionaryEntry
        // list.

        if (word_type == word_desc->type)
            switch (word_type)
            {
                case article:
                    int temp;
                    #ifdef DEBUG
                    cerr << "\t\t\t\t\tENTRY Dictionary::get_word looking for article" << endl;
                    #endif

                    if (word_index == word_desc->word_index) // Word found
                        return 1;
                    else // Not found
                        return 0;
            }
```
# dictionary/src/dictionary.C

```c
130  #endif
131  #endif  // search verb list for word index
132  // match, then ask DictionaryEntry for the
133  // necessary derivation.
134  //-----------------------------
135  for(DictionaryEntry_verb * ptr3 = verbs;
136         ptr3->get_index() == word_index;ptr3++)
137      #ifdef DEBUG
138               cerr << "\t\t\tDEBUG Dictionary::get_word looking for verb" << end;
139      #endif
140      //cerr << ptr3->get_index();
141      cerr << "\t\t\tlooking for ";
142      cerr << word_index;
143      cerr << "\t\t\t" << end;
144      #endif
145  return ptr3->derive_word(word_desc);
146  case adjective:
147      #ifdef DEBUG
148               cerr << "\t\t\tDEBUG Dictionary::get_word looking for adjective" << end;
149      #endif
150  //-----------------------------
151  for(DictionaryEntry_adjective * ptr4 = adja;
152         ptr4->get_index() == word_index;ptr4++)
153      #ifdef DEBUG
154               cerr << "\t\t\tDEBUG Dictionary::get_word loops and finds index"
155      #endif
156               cerr << ptr4->get_index();
157               cerr << "\t\t\t(looking for ";
158               cerr << word_index;
159               cerr << "\t\t\t" << end;
160      #endif
161  return ptr4->derive_word(word_desc);
162  case UD_UNKNOWN_TYPE:
163      cerr << "\t\t\tERROR Unknown word type passed to dict" << end;
164      #ifdef DEBUG
165               cout << "\t\t\tERROR Unknown word type passed to dict" << end;
166      #endif
167  default:
168      break;
169  }
170  #endif
171  #ifdef DEBUG
172               cout << "\t\t\tENTRY Dictionary::get_level" << end;
173      #endif
174  #ifdef DEBUG
175               cout << "\t\t\tDEBUG input word index is " << word_index << end;
176      #endif
193  //--------- Automatics - objects ---------
194  //--------- Automatics - other ---------
195  #ifdef DEBUG
196  cout << "\t\t\tERROR - Invalid word index" << end;
197      #endif
```
```c
// File: dictionaryentry.h

// Base class

DictionaryEntry::DictionaryEntry(int init_index, int init_level)
{
    // Initialize index and level
    index = init_index;
    knowledge_level = init_level;
}

DictionaryEntry::~DictionaryEntry()
{
    // Destructor
}

int DictionaryEntry::get_index()
{
    // Return index
    return index;
}

int DictionaryEntry::get_level()
{
    // Return knowledge level
    return knowledge_level;
}

DictionaryEntry::DictionaryEntry(const DictionaryEntry &other)
{
    // Copy constructor
    index = other.index;
    knowledge_level = other.knowledge_level;
}

DictionaryEntry &DictionaryEntry::operator=(const DictionaryEntry &other)
{
    // Assignment operator
    if (this != &other)
    {
        index = other.index;
        knowledge_level = other.knowledge_level;
    }
    return *this;
}

DictionaryEntry::DictionaryEntry(const DictionaryEntry &other, int new_level)
{
    // Copy constructor with level
    index = other.index;
    knowledge_level = new_level;
}

DictionaryEntry::DictionaryEntry(const DictionaryEntry &other, int new_index, int new_level)
{
    // Copy constructor with index and level
    index = other.index;
    knowledge_level = new_level;
}

DictionaryEntry::DictionaryEntry(const DictionaryEntry &other, int new_index, int new_level, int new_knowledge)
{
    // Copy constructor with index, level, and knowledge
    index = other.index;
    knowledge_level = new_knowledge;
}

DictionaryEntry::DictionaryEntry(const DictionaryEntry &other, int new_index, int new_level, int new_knowledge, int new_word)
{
    // Copy constructor with index, level, knowledge, and word
    index = other.index;
    knowledge_level = new_knowledge;
}

```

```c
#include "ENTRY.h"

#define DEBUG 1

ENTRY::ENTRY(int init_index, int init_know_level, char *init_g_sing, char *init_g_plural, char *init_e_sing, char *init_e_plural)
{
  if (DEBUG) { cerr << "ENTRY Constructor" << endl; }
  Gender init_gend;
  gender = init_gend;
  g_sing = init_g_sing;
  g_plural = init_g_plural;
  e_sing = init_e_sing;
  e_plural = init_e_plural;
}
```

```c
ENTRY::ENTRY(int init_index, int init_know_level)
{
  if (DEBUG) { cerr << "ENTRY Constructor" << endl; }
  Gender init_gend;
  gender = init_gend;
  g_sing = init_g_sing;
  g_plural = init_g_plural;
  e_sing = init_e_sing;
  e_plural = init_e_plural;
}
```

```c
 ENTRY::ENTRY(int init_index)
{
  if (DEBUG) { cerr << "ENTRY Constructor" << endl; }
  Gender init_gend;
  gender = init_gend;
  g_sing = init_g_sing;
  g_plural = init_g_plural;
  e_sing = init_e_sing;
  e_plural = init_e_plural;
}
```

269 )
269  |
270  |
271  |
272  |
273  |
274  |
275  |
276  |
277  |
278  |
279  | Derive class -----------------------------/
280  | Constructor.-----------------------------/
281  | Derive class -----------------------------/
282  | Constructor.-----------------------------/
283  | DictionaryEntry_adjective::DictionaryEntry_adjective(int init_index,
284  | char * init_e, char * init_g) :
285  | DictionaryEntry::DictionaryEntry(init_index, init_know_level)
286  | #ifdef DEBUG
287  | cerr << "\t\t\t\tENTRY DictionaryEntry_adjective CONSTRUCTOR" << endl;
288  | #endif
289  |
290  |
291  | Derr = init_e;
292  | ger = init_g;
293  |
294  |
295  |
296  |
297  |
298  |
299  | //-----------------------------------------
300  | // Destructor.
301  | //-----------------------------------------
302  | DictionaryEntry_adjective::DictionaryEntry_adjective()
303  | #ifdef DEBUG
304  | cerr << "\t\t\t\tENTRY DictionaryEntry_adjective DESTRUCTOR" << endl;
305  | #endif
306  |
307  |
308  |
309  |
310  |
311  |
312  |
313  | Word DictionaryEntry_adjective::derive_word(WordDescriptor * word_desc)
314  | #ifdef DEBUG
315  | cerr << "\t\t\t\tENTRY DictionaryEntry_adjective::derive_word" << endl;
316  | #endif
317  |
318  |
319  |
320  |
321  |
322  |
323  |
324  |
325  |
326  |
327  | derived_word.set_index(DictionaryEntry::get_index());
328  | derived_word.set_type(adjective);
329  | derived_word.set_language(word_desc->get_language());
330  | Underived word set (None, G_english, G_german, G_french, G_italian,
331  | derived_word.set_count(G_none);
derived_word.set_gender(G_none);
if (word_desc->get_language() == L_english)
    derived_word.set_string(eng);
else
    derived_word.set_string(ger);
return(derived_word);
133 Language DynamicWordDescriptor::get_language()
134 #ifdef DEBUG
135     cerr << "\t\t\tENTRY DynamicW::D:::get_language" << endl;
136 #endif
137     //--------------  Automatics - objects --------------/
138     //--------------  Automatics - other --------------/
139     //--------------  Code begins ---------------------/
140     return(language);
141     //--------------  Code ends ----------------------/

```c
// 04-19-93 session/src/exit_tutor.C Page 1

#include <iostream.h>
#include <stdlib.h>
#include "log.h"
#include "statistics.h"
#include "userlist.h"

extern Log log;
extern Statistics stats;
extern time_t session_start_time;
extern UserList users;

void exit_tutor()
{
    // Automatics - objects
    // Automatics - other
    // Print a final statistics summary for the
    // user.
    stats.print_summary();
}
```

```c
May 9 1993 19:12:32
```

```c
66     // Log final statistics, and write the log
67     // file.
68     log.final_stats(stats);
69     log.save();
70     // Save the user data file.
71
72     users.write();
73
74     // Say goodbye...
75
76     cout << "The session with the German Language Tutor is complete." << endl;
77     cout << endl;
78     exit(0);
```
// Filename: hint.C
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file implements the Hint class.
// Notes: The theory and design of this system have been documented in the accompanying thesis report.
// Revisions By Reason
// ------- -------
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#include <iostream.h>
#include <stdlib.h>

Hint::Hint(char * in_hint) {
    ifdef DEBUG
    cerr << "\t\t\t\tENTRY Hint CONSTRUCTOR" << endl;
    endif
    hint = in_hint;
}

Hint::~Hint() {
    ifdef DEBUG
    cerr << "\t\t\t\tENTRY Hint DESTRUCTOR" << endl;
    endif
}

void Hint::print() {
    ifdef DEBUG
    cerr << "\t\t\t\tENTRY Hint::print" << endl;
    endif
    //---------- Automatics - objects ----------/
    //---------- Automatics - other ----------/
    //---------- Code begins ----------/
    cout << endl;
// Filename: log.C
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This file implements the run-time Log class.
// Notes: The theory and design of this system have been documented in the accompanying thesis report.

#include <iostream>
#include <fstream.h>
#include <string.h>
#include <iomanip.h>
#include <iostream.h>
#include "logentry.h"
#include "statistics.h"
#include "word_end_phrase_defs.h"
#include "statistic_defs.h"
#include "userinterface.h"
#include "userinterface_defs.h"

// External data
extern UserInterface userint;

class Log:

public:

Log():Log()
{
    // Constructor.
}

Log::Log()
{
    #ifdef DEBUG
    cerr << "\t<ENTRY Log CONSTRUCTOR* " << endl;
    #endif
    current_username = NULL;
    current presentation phrase = NULL;
    current_entry_count = 0;
    current_model_index = 0;
    list = new LogEntry *[MAX_CURRENT_LOG_ENTRIES];
}

// Destructor.

Log::~Log()
{
    #ifdef DEBUG
    cerr << "\t\t<ENTRY Log DESTRUCTOR* " << endl;
    #endif
    for (int i = 0; i < current_entry_count; i++)
```c
#include "log.h"

void Log::final_stats(Statistics in_stats)
{
  // Member function set_current_user()
  // --
  // Create a new LogEntry object and add it to
  // the current list.

  entry_ptr = new LogEntry_final_stats(current_username, in_stats);
  list[current_entry_count++] = entry_ptr;
}

void Log::save()
{
  // Member function set_current_user()
  // --
  // Create a new LogEntry object and add it to
  // the current list.

  ofstream outFile;
  // and request that each print itself to the
  // log file.

  outFile.open("gltr.log", ios::app);
  if (!outFile)
    cerr << "ERROR cannot open gltr.log for append" << endl;
  else
    // Loop through the current list of LogEntries, //
    // and print each LogEntry to the
    // log file.

    for (int i=0; i<current_entry_count; i++)
      list[i]->print(&outFile);
  
  outFile.close();
  return;
}
```
log/src/log.C

264  //---------- Automatons - other ----------//
265  char ch;
266  int int_input;
267  int next_entry_type;
268  LogEntry* LE_at_leaf;
269  LogEntry* LE_entry;
270  LogEntry_final_stats* LE_final_stats;
271
272  //---------- Code begins ----------
273
274  //--------- Log file management ---------
275  // Open the log file for input. Read the
276  // Log entry type for each entry, create an
277  // appropriate LogEntry object, and ask it
278  // to read its data.
279
280  //---------- Code ends ----------
281
282  log_stream.open("gtt.log", ios::in);
283  if (!log_stream)
284  { cerr << "ERROR - could not open log file." << endl;
285   return 1;
286  }
287  while (log_stream.get(ch))
288  {
289    log_stream.putback(ch);
290    log_stream >> int_input;
291    log_stream >> next_entry_type;
292    switch(next_entry_type)
293    {
294  case LE_at_leaf:
295    LE_at_leaf = new LogEntry_LE_at_leaf;
296    break;
297    case LE_final_stats:
298    LE_final_stats = new LogEntry_LE_final_stats;
299    break;
300    case LE_UNKNOWN_TYPE:
301    default:
302      cerr << "ERROR: Unknown log entry type read." << endl;
303      break;
304    }
305  log_stream.close();
306  }
307
308  //---------- View at leaves ----------
309  void Log::view_at_leaf();
310  {
311    if (DEBUG) cerr << "View at leaves" << endl;
312    #endif
313    }
ControlCommand choice;

if (list[i]->get_type() == LS_FINAL_STATS) // If we're found, display that.
c_out << "No further statistics entries found." << endl;
c_out << "End of log." << endl;
}
else if (choice != CC_admin_find_next) //
  cerr << "ERROR: bad choice returned" << endl;
}

if (found) //
  if (choice != CC_quit) //
    cout << endl;
    cout << "Cumulative statistics from log file." << endl;
    cout << "\tNumber of individual sessions counted - " << num_stats << endl;
  int_val = int_val / NUM_SECONDS_PER_HOUR;
  hours = int_val / NUM_SECONDS_PER_MINUTE;
  minutes = int_val - hours * NUM_SECONDS_PER_MINUTE;
  seconds = int_val - minutes * NUM_SECONDS_PER_MINUTE;
  cout << "\tTotal duration of sessions = ";
  if (hours < 1) //
    cout << minutes << " minutes, " << seconds << " seconds." << endl;
  else //
    cout << hours << " hours, " << minutes << " minutes." << endl;

// This function is called from the main menu.
void Log::view_cum_stats()
{
  int Val = 0;
  hours = int_val / NUM_SECONDS_PER_HOUR;
  minutes = int_val / NUM_SECONDS_PER_MINUTE;
  seconds = int_val - minutes * NUM_SECONDS_PER_MINUTE;
  cout << "\tTotal duration of sessions = ";
  if (hours < 1) //
    cout << minutes << " minutes, " << seconds << " seconds." << endl;
  else //
    cout << hours << " hours, " << minutes << " minutes." << endl;
}

// This function is called from the main menu.
LogEntry::view_cum_stats()
{
  int Val = 0;
  hours = int_val / NUM_SECONDS_PER_HOUR;
  minutes = int_val / NUM_SECONDS_PER_MINUTE;
  seconds = int_val - minutes * NUM_SECONDS_PER_MINUTE;
  cout << "\tTotal duration of sessions = ";
  if (hours < 1) //
    cout << minutes << " minutes, " << seconds << " seconds." << endl;
  else //
    cout << hours << " hours, " << minutes << " minutes." << endl;
}

// This function is called from the main menu.
Log::view_current_stats()
{
  int Val = 0;
  hours = int_val / NUM_SECONDS_PER_HOUR;
  minutes = int_val / NUM_SECONDS_PER_MINUTE;
  seconds = int_val - minutes * NUM_SECONDS_PER_MINUTE;
  cout << "\tTotal duration of sessions = ";
  if (hours < 1) //
    cout << minutes << " minutes, " << seconds << " seconds." << endl;
  else //
    cout << hours << " hours, " << minutes << " minutes." << endl;
}
int_val = int_val - (hours * NUM_SECONDS_PER_HOUR);
minutes = int_val/NUM_SECONDS_PER_MINUTE;
seconds = int_val % NUM_SECONDS_PER_MINUTE;

if (hours < 1)
    cout << "Average duration of sessions - ";
    cout << minutes << " minutes, " << seconds << " seconds." << endl;
else
    cout << hours << " hours, " << minutes << " minutes." << endl;

cout << "Total number of phrases presented by system - " << cum_exerc
     se_count << endl;
cout << "Total number of student responses - " << cum_attempt_count <<
     endl;
cout << "Total number of correct translations - " << cum_correct_count
     << endl;

choice = userint.after_cum();
if (choice == CC_quit)
    { cerr << "ERROR: bad choice returned" << endl;
    }
May 9 1993 19:12:32 log/src/logentry.C Page 1

1  //
2  //  Filename: logentry.C
3  //
4  //  Project: German Language Tutor, RIT Master's Thesis
5  //
6  //  Purpose: This file implements the LogEntry class and its derived
7  //           classes.
8  //
9  //  Notes: The theory and design of this system have been documented
10  //         in the accompanying thesis report.
11  //
12  //  Revisions By Reason
13  //  --------- -------
15  //
16  //-------------- Included files ------------------/
17
18  #include <iostream.h>
19  #include <fstream.h>
20  #include <string.h>
21  #include <time.h>
22  #include "logentry.h"
23  #include "statistics.h"
24  #include "word_and_phrase_defs.h"
25
26  //-------------- Base class ---------------------/
27  // Constructor 1 of 2.
28  //
29  LogEntry::LogEntry()
30  {
31  #ifdef DEBUG
32      cerr << "\t\t\t\t\tENTRY LogEntry CONSTRUCTOR 1 of 2" << endl;
33  #endif
34  
35  // Expect that internal data will be set by a derived class read() or
36  // Constructor 2 of 2.
37  //
38  
39  LogEntry::LogEntry(char * in_username, int in_index,
40                     LE_type in_type)
41  {
42  #ifdef DEBUG
43      cerr << "\t\t\t\t\tENTRY LogEntry CONSTRUCTOR 2 of 2" << endl;
44  #endif
45
46  // Set internal data based on input, and stamp the new entry with the current
47  // date and time.
48  //
49  //-------------- Derived class ---------------------/
50
51  LogEntry_at_leaf::LogEntry_at_leaf(LE_type in_type)
52  {
53  #ifdef DEBUG
54      cerr << "\t\t\t\t\tENTRY LogEntry_at_leaf CONSTRUCTOR 1 of 2" << endl;
55  #endif
56
57  LogEntry_at_leaf::LogEntry_at_leaf(char * in_username, int in_index,
58                                        LE_at_leaf_type in_type, Phrase in_exp_response,
59                                        Phrase in_user_response) : LogEntry(in_username, in_index, LE_at_leaf)
60  {
61  #ifdef DEBUG
62      cerr << "\t\t\t\t\tENTRY LogEntry_at_leaf CONSTRUCTOR 2 of 2" << endl;
63  #endif
64

66  time_t tloc;
67
68  username = new char[sizeof(in_username)];
69  strcpy(username, in_username);
70  model_index = in_index;
71  record_type = in_type;
72  date_time_stamp = time(&tloc);
73  
74  LogEntry :: LogEntry()
75  {
76      #ifdef DEBUG
77          cerr << "\t\t\t\t\tENTRY LogEntry DESTRUCTOR" << endl;
78      #endif
79      
80
81  LE_type LogEntry::get_type()
82  {
83      #ifdef DEBUG
84          cerr << "\t\t\t\t\tENTRY LogEntry::get_type" << endl;
85      #endif
86
87        return(record_type);
88  
89  LE_type LogEntry_at_leaf::get_type()
90  {
91      #ifdef DEBUG
92          cerr << "\t\t\t\t\tENTRY LogEntry_at_leaf::get_type" << endl;
93      #endif
94
95      
96
```c
#define DEBUG

cerr << "\DEBUG expected response is " << expected_response << endl;
cerr << "\DEBUG actual response is " << user_response << endl;
cerr << "\DEBUG presentation phrase is " << presentation_phrase << endl;
}
#endif DEBUG

type = in_type;
}

LogEntry_at_leaf::LogEntry_at_leaf()
{
  #ifdef DEBUG
  cerr << "\ENTRY LogEntry_at_leaf DESTRUCTOR" << endl;
  #endif

delete [] presentation_phrase;
delete [] expected_response;
delete [] user_response;
}

// Destructor.

LogEntry_at_leaf::LogEntry_at_leaf(const LogEntry_at_leaf& rhs)
{
  // Copy constructor.

delete [] presentation_phrase;
delete [] expected_response;
delete [] user_response;
}

_LOGENTRY торговой лавки: LogEntry_at_leaf

void LogEntry_at_leaf::print(const char* message)
{
  // Print message.

  LOGENTRY торговой лавки: print" << endl;

  // Print object data.

  *log_stream << record_type << endl;
  *log_stream << type << endl;
  *log_stream << user_name << endl;
  *log_stream << date_time_stamp << endl;
  *log_stream << model_index << endl;
  *log_stream << presentation_phrase << endl;
  *log_stream << expected_response << endl;
  *log_stream << user_response << endl;
}
```
input_line[ch_cnt-1] = '\0';
264
presentation_phrase = new char[ch_cnt];
265
strcpy(presentation_phrase, input_line);
266
end_of_line = FALSE;
267
for (ch_cnt = 0; end_of_line && log_stream->get(in_ch); ch_cnt++)
268            if (in_ch == '\n')
269                end_of_line = TRUE;
270            else
271                input_line[ch_cnt] = in_ch;
272
            }
273
input_line[ch_cnt-1] = '\0';
274
expected_response = new char[ch_cnt];
275
strcpy(expected_response, input_line);
276
end_of_line = FALSE;
277
for (ch_cnt = 0; end_of_line && log_stream->get(in_ch); ch_cnt++)
278            if (in_ch == '\n')
279                end_of_line = TRUE;
280            }
281
            input_line[ch_cnt] = in_ch;
282
            }
283
}
284
if (type == LE_at_leaf_hint)
285            cout << "Leaf node encountered while looking for additional hints." << endl;
286        }
287    }
288    else
289        cout << "Leaf node encountered while trying to diagnose." << endl;
290    }
291
    // Format the date/time stamp, and output all
    // entry data to the specified log file.
    //---------------------------------------------
    my_ptr = localtime(&date_time_stamp);
    void strftime( formatted_date_time_stamp, MAX_DATE_STAMP_SIZE, "%a %b %d %Y",
    my_ptr );
294
    cout << "\tDate - " << formatted_date_time_stamp << " Username - "
295            << name << endl;
296        cout << "\tIndex of model chosen - " << model_index << endl;
297        cout << "\tPhrase which was to be translated - "
298                << presentation_phrase << endl;
299        cout << "\tExpected response - "
300                << expected_response << endl;
301        cout << "\tActual response - " << user_response << endl;
302        cout << endl;
303    }
304
    //--------------------------------------------- Derived class -----------------------------/
305
    LogEntry_final_stats::LogEntry_final_stats(LE_type in_type)
306    {
    #ifdef DEBUG
307            cerr << "\t\t\t\t\tENTRY LogEntry_final_stats CONSTRUCTOR 1 of 2" << endl;
308            cerr << "\t\t\t\t\tENTRY (for automatics to be read)" << endl;
309            #endif
310            record_type = in_type;
311        }
312        
313        void LogEntry_at_leaf::display()
314            {
            cerr << "\t\t\t\t\tENTRY LogEntry_at_leaf::display" << endl;
315        
    #ifdef DEBUG
316            cerr << "\t\t\t\t\tENTRY LogEntry_final_stats CONSTRUCTOR 2 of 2" << endl;
317            cerr << "\t\t\t\t\tENTRY (for run-time created entries)" << endl;
            #endif
318        
319        //------ Code begins --------------------------/
320        
321        struct tm * my_ptr;
322        char formatted_date_time_stamp[MAX_DATE_STAMP_SIZE];
323
    //------ Code ends --------------------------/


```c
if (in_ch == \n"

```
May 9, 1993 19:12:32
log/src/logentry.C
Page 9

120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140

// Member function get_stats
//

stats::stats() { }
```c
#define DEBUG

cerr << "\t\t\t\ENTRY PatternNode:constructor" << endl;
#endif

PatternNode::PatternNode(Pattern *in_func, PatternNode *in_mat, 
    PatternNode *in_nomat, Hint *in_hint)
{
    #ifdef DEBUG
    hint << "\t\t\ENTRY PatternNode CONSTRUCTOR" << endl;
    #endif

    hint = in_hint;
    pattern_constructor = in_func;
    match_path_pattern = in_mat;
    no_match_path_pattern = in_nomat;
);

    // Constructor.
    // Constructor.
    // Constructor.
    // Constructor.
    // Constructor.
    // Constructor.

    PatternNode::~PatternNode()
    {
    #ifdef DEBUG
    cerr << "\t\t\ENTRY PatternNode DESTRUCTOR" << endl;
    #endif

    }

    // Destructor.
    // Destructor.
    // Destructor.
    // Destructor.
    // Destructor.
    // Destructor.

    boolean PatternNode::process_response(Phrase expected_response, 
        Phrase user_response)

    {
```


```c
    #ifdef DEBUG
    cerr << "\t\t\t\ENTRY PatternNode:process_response()" << endl;
    #endif

    //--------------- Automatics - objects ---------------

    Pattern wildcard_pattern = (*pattern_constructor)(expected_response);

    //--------------- Automatics - other ---------------

    ControlCommand choice1;
    boolean try_again;

    //--------------- Code Begins ---------------

    // The first thing PatternNode does to
    // process a user response is to attempt
    // to match it against its own generated
    // Pattern.

    if (wildcard_pattern.match(user_response))
    {
        // if the Pattern matched, it means that an
        // incorrect response has been recognized.
        // Offer the user options.

        choice1 = userint.wrong_answer();

        #ifdef DEBUG
        cerr << "\t\t\ENTRY pattern required new action. Construction" << endl;
        #endif
        try_again = FALSE;
    }
    }
    }
    }
    }
    }
    }
```

May 9 1993 19:12:32 models/src/patternnode.C Page 1

```c
    #ifdef DEBUG
    cerr << "\t\t\ENTRY PatternNode:process_response()" << endl;
    #endif

    //--------------- Automatics - objects ---------------

    Pattern wildcard_pattern = (*pattern_constructor)(expected_response);

    //--------------- Automatics - other ---------------

    ControlCommand choice1;
    boolean try_again;

    //--------------- Code Begins ---------------

    // The first thing PatternNode does to
    // process a user response is to attempt
    // to match it against its own generated
    // Pattern.

    if (wildcard_pattern.match(user_response))
    {
        // if the Pattern matched, it means that an
        // incorrect response has been recognized.
        // Offer the user options.

        choice1 = userint.wrong_answer();

        #ifdef DEBUG
        cerr << "\t\t\ENTRY pattern required new action. Construction" << endl;
        #endif
        try_again = FALSE;
    }
    }
    }
    }
    }
    }
```
// to give more detailed hints.            
//--------------------------------------
if (match_path_pattern == NULL) {
    log_at_leaf(LF_at_leaf_hint, expected_response, 
    user_response);
    choice = user_int.cant_diagnose_further();
    if (choice == CC_Try_Again)
        try_again = TRUE;
    else
        try_again = FALSE;
} else
    try_again = match_path_pattern->more_hints(
        expected_response, user_response);
    if (try_again)
        try_again = TRUE;
    else
        try_again = FALSE;
} else if (choice == CC_Correct_Answer)
    try_again = FALSE;
} else
    try_again = TRUE;
}

#define DEBUG
if (wildcard_pattern.match(user_response)) {
    // if the Pattern matched, it means that the 
    // problem with the response has been further 
    // isolated.  Pass this Pattern's hint to the 
    // Userinterface object for printing, and 
    // additional user options.
    choice = User_int.after_hint(hint);
    if (choice == CC_Hint) 
        // if the user requests an additional hint, 
        // try to move down the Match Path for more 
        // detailed diagnosis.  If this is not 
        // possible, make a log entry indicating that 
        // a leaf node has been hit while attempting 
        // to give more detailed hints.
    if (match_path_pattern == NULL) 
        log_at_leaf(LF_at_leaf_diag, expected_response, 
        user_response);
        choice = user_int.cant_diagnose();
    if (choice == CC_Try_Again)
        try_again = TRUE;
log.at_leaf(LX_at_leaf_hint, expected_response, user_response);
choice1 = userint.cant_diagnose_further();
if (choice1 == CC_try_again)
    try_again = TRUE;
else
    try_again = FALSE;
else if (choice1==CC_correct_answer)
    try_again = FALSE;
else
    try_again = TRUE;
else
    try_again = FALSE;

log.at_leaf(LX_at_leaf_hint, expected_response, user_response);
choice1 = userint.cant_diagnose_further();
if (choice1 == CC_try_again)
    try_again = TRUE;
else
    try_again = FALSE;
else
    try_again = FALSE;

if (no_match_path_pattern == NULL)
    log.at_leaf(LX_at_leaf_hint, expected_response, user_response);
choice1 = userint.cant_diagnose_further();
if (choice1 == CC_try_again)
    try_again = TRUE;
else
    try_again = FALSE;

try_again = FALSE;
else
    try_again = TRUE;

try_again = no_match_path_pattern->more_hints(
    expected_response, user_response);
else
    return(try_again);
---

// File: phrase_descriptor.C

#include <iostream.h>
#include "word.h"
#include "dictionary.h"
#include "phrase_descriptor.h"
#include "dynamic_word_descriptor.h"

// External data

extern Dictionary dictionary;

// Constructor.

PhraseDescriptor::PhraseDescriptor(int list_size, int * in_order,
    WordDescriptor * wd1,
    WordDescriptor * wd2, WordDescriptor * wd3,
    WordDescriptor * wd4, WordDescriptor * wd5,
    WordDescriptor * wd6, WordDescriptor * wd7,
    WordDescriptor * wd8, WordDescriptor * wd9,
    WordDescriptor * wd10)
{
    #ifdef DEBUG
        cerr << "\t\tENTRY PhraseDescriptor CONSTRUCTOR* " << endl;
    #endif

    // Create any array which will hold pointers
    // to the individual WordDescriptors.

    descriptors = new WordDescriptor * [list_size];

    size = list_size;
    fillin_order = in_order;

    // Get any descriptors which were specified.

    switch (list_size)
    {
    case 10:
        descriptors[9] = wd10;
    case 9:
#construction/src/phrasedescriptor.C

```c
131  cerr << "\t\t\DEBUG Echoing the word indices for the" << endl;
132  cerr << "\t\t\DEBUG BasePhrase input to _Di:build_phrase." << endl;
133  int temp = _in_bp.get_index(1);
134  temp = _in_bp.get_index(2);
135  temp = _in_bp.get_index(3);
136  temp = _in_bp.get_index(4);
137  temp = _in_bp.get_index(5);
138 #endif
139 // ------------------------------------------------------------------------
140 // Loop based on the size of the Phrase //
141 // Descriptor, but process each word based //
142 // on the fill-in order (necessary to ensure //
143 // that words which must agree with each other //
144 // are obtained in the proper order). //
145 // ------------------------------------------------------------------------
146 for (int size_count = 0; size_count < size; size_count++)
147 { 
148    order_index = fillin_order[size_count];
149 #ifdef DEBUG
150    cerr << "\t\t\DEBUG Build phrase order index used is ";
151    cerr << order_index << endl;
152 #endif
153 // ------------------------------------------------------------------------
154 // Get the appropriate word_index and Word //
155 // Descriptor - the two things needed in order //
156 // to obtain an actual word from the dictionary. //
157 // ------------------------------------------------------------------------
158    word_index = _in_bp.get_index(order_index);
159 #ifdef DEBUG
160    cerr << "\t\t\DEBUG got word index " << word_index << " get WD. " << endl;
161 #endif
162    WordDescriptor word_desc(descriptors[order_index-1]);
163 // ------------------------------------------------------------------------
164 // The current WordDescriptor may contain //
165 // attributes which are dynamic (held over to //
166 // agree with earlier words). While this is //
167 // true, loop back through already processed //
168 // words until all dynamic attributes are //
169 // resolved. //
170 // ------------------------------------------------------------------------
171 #ifdef DEBUG
172    cerr << "\t\t\DEBUG got WD. work on dynamics" << endl;
173 #endif
174    DynamicWordDescriptor dyna WD;
175    for (int past_cnt = size_count-1; past_cnt >= 0 &
176    !Word_desc.no_dynamics(); past_cnt--)
177     [ dyna WD.resolve_dynamics(words[fillin_order[past cnt]-1]);
178      word_desc.resolve_dynamics(dyna WD);
179     ]
180 // ------------------------------------------------------------------------
181 // Get the current word from the dictionary, //
182 // and store it. //
183 // ------------------------------------------------------------------------
184    words[order_index-1] = dictionary.get_word(word_index, word_desc);
185 }```

```c

    if (knowledge_level <= in_level)
        if (phrases->reset_selections(in_level) == TRUE)
            if (DEBUG) cerr << "\t\t\t\tDEBUG BPP reports it has active lists" << endl;
        #endif
        return(TRUE);
    else
        #ifdef DEBUG
            cerr << "\t\t\t\tDEBUG BPP reports it has active lists" << endl;
        #endif
```
```c
void PhraseModelList::do_exercise()
{
    if (model_list().reset_depletions[current_level] == TRUE)
    {
        depleted[i] = FALSE;
        number_active_models++;
    } else
    {
        depleted[i] = TRUE;
    }
    exercise_done = this->find_and_try();
}
```

```c
    if (!exercise_done)
    {
        for (int i = 0; i < size; i++)
        {
            if (model_list().reset_depletions[current_level] == TRUE)
            {
                depleted[i] = FALSE;
                number_active_models++;
            } else
            {
                depleted[i] = TRUE;
            }
            exercise_done = this->find_and_try();
        }
    }
```

```c
    if (!exercise_done)
    {
        for (int i = 0; i < size; i++)
        {
            if (model_list().reset_depletions[current_level] == TRUE)
            {
                depleted[i] = FALSE;
                number_active_models++;
            } else
            {
                depleted[i] = TRUE;
            }
            exercise_done = this->find_and_try();
        }
    }
```
int num_undepleted = 0;
int rand_val;
int chosen_count;
int chosen_index;

//----------- Code begins -----------------------------

// Determine how many PhraseModels have not yet
// exhausted all of their vocabulary selections.
for (int i = 0; i < size; i++)
{
    if (depleted[i] == FALSE)
    {
        num_undepleted++;
    }

// While an exercise has not yet been done,
// try the different PhraseModels.
while (still_trying)
{
    if (num_undepleted == 0)
    {
        still_trying = FALSE;
    }
    else
    {
        // Get a pseudo-random number in the range of
        // undepleted PhraseModels.
        rand_val = rand();
        chosen_count = rand_val % num_undepleted;

        //DEBUG number of undepleted entries is
        //rand_val << num_undepleted;
        //DEBUG random value obtained is
        //chosen_count = modulo 2 end is
        //chose
        //undepleted PhraseModels.
        //Use the chosen random index to step into
        //the undepleted PhraseModels.
        for (chosen_index = 0; chosen_count > 0; chosen_index++)
        {
            if (depleted[chosen_index] == FALSE)
            {
                chosen_count--;
            }
        }
    }
}

log.set_current_model_index(chosen_index);
exercise_done = model_list[chosen_index].do_exercise();
if (exercise_done)
{
    stilltrying = FALSE;
}
else
{
    num_undepleted--;
    depleted[chosen_index] = TRUE;
    stilltrying = TRUE;
}

return (exercise_done);

// Member function set_current_knowledge_level.

void PhraseModelList::set_current_knowledge_level(int in_level) {
    ifdef DEBUG
    cerror("\t\t\t\t\tENTRY PhraseModelList::set_cur_know_level " << endi;
    endif

    int number_active_models = 0;

    // Code begins
    // Loop through list, asking each PhraseModel
    // to set its knowledge level to match the
    // current user's.
    // CURRENT_LEVEL = in_level;
    for (int i = 0; i < size; i++)
    {
        if (model_list[i].set_level(current_level) == TRUE)
        {
            depleted[i] = FALSE;
            number_active_models++;
        }
    }
    else
    {
        depleted[i] = TRUE;
    }
}
If you are reading this comment, you are
probably looking at the code in some amount of detail. You
should be warned that I was learning the language as I used
it - not the ideal situation for the program which is
being written! As I learned C++ better, I did go back and
correct some glaring errors in usage, which no doubt would
have manifested themselves as subtle bugs later. However,
I am sure I didn't get them all, not to mention the myriad
style issues which could not even begin to address given
time constraints. I would really liked to have gone back
and reviewed and updated the code strictly for the use of
const and references, copy constructors, operators, etc. You
will probably run into some ugly code! If you are planning
to enhance, or otherwise use this program, you should be
aware that is very much a prototype, and it would be
wise to look at the thesis report, especially the
conclusions chapter. I may also have made enhancements
myself (just for fun :-), so feel free to drop me a line
(assuming I am still at the same e-mail address) to ask.

Good luck!

Last known e-mail address: staffan@serum.kodak.com.

---

### Included files

```
#include <iostream.h>
#include "dictionary.h"
#include "stdtype.h"
#include "session.h"
#include "phraseModellist.h"
#include <stdlib.h>
#include <sys/time.h>
#include "statistics.h"
#include "userinterface.h"
#include "log.h"
#include "userlist.h"
```

typedef long time_t;

### Global Data

```
UserInterface userint;
Log log;
```
long_seed = time(tloc);

```c
#ifdef DEBUG
cerr << "\n\nDEBUG long seed value of " << long_seed << endl;
#endif
```

```c
int_seed = (unsigned int) long_seed;

#ifdef DEBUG
cerr << "\n\nDEBUG int seed value of " << int_seed << endl;
#endif
```

```c
switch (user_type)
{
    case STUDENT:
        //
        // If student user, set current knowledge
        // level and begin exercises.
        //
        #ifdef DEBUG
cerr << "\n\nDEBUG taking action for STUDENT" << endl;
#endif
        user_wants_to_continue = TRUE;
exercise_count = 0;

        cout << endl;
cout << "\nYou will now be asked to translate various phrases and"
<< " sentences. You may stop at any time - the important com"
<< " and" << endl;
cout << "\nto remember is \"\" Type a question mark at anytime to "
<< endl;
cout << "\nbe told your options and be given the opportunity to lea"
<< endl;
cout << endl;
models.set_current_knowledge_level(6);

    while (user_wants_to_continue)
    {
        #ifdef DEBUG
cerr << "\n\nDEBUG invoking models to do an exercise" << endl;
#endif
```
```c
stats.print_summary();
cout << endl;
choice = userint.after_stats();
if (choice == CC_Try_again)
    user_wants_to_continue = TRUE;
else
    user_wants_to_continue = FALSE;

// user_wants_to_continue = TRUE;
// end while
break;

// case SYSTEM:
#elif !defined DEBUG
    err << "\"\"Taking action for system designer\"\" << endl;
#endif

//System administrator features.

log.load();
// test code append it back - file should double.
log.save();
for(;;)
{
    choice = userint.sysadmin();
    switch(choice)
    {
        case CC_admin_write:
            cout << "The system administrator command to write*" << endl;
            cout << "current log entries to the default log file has" << endl;
            break;
        case CC_admin_delete:
            cout << "The system administrator command to delete the*" << endl;
            cout << "current log file has not yet been enabled." << endl;
            break;
        case CC_admin_save:
            cout << "The system administrator command to save the current" << endl;
            cout << "log file under a new name has not yet been enabled." << endl;
            break;
        case CC_admin_diag_fail:
            log_view_at_leaf();
            break;
        case CC_admin_report_diag_fail:
            cout << "The system administrator command to generate a*" << endl;
```
void Statistics::increment_attempt_count()
{
    if (DEBUG) {
        cerr << \"\"ENTRY Statistics::increment_attempt_count\" << endl;
    }
    attempt_count++;
    return;
}

void Statistics::increment_correct_count()
{
    if (DEBUG) {
        cerr << \"\"ENTRY Statistics::increment_correct_count\" << endl;
    }
    correct_count++;
    return;
}

void Statistics::print_summary()
{
    if (DEBUG) {
        cerr << \"\"ENTRY Statistics::print_summary\" << endl;
    }
    return;
}

void Statistics::increment_exercise_count()
{
    if (DEBUG) {
        cerr << \"\"ENTRY Statistics::increment_exercise_count\" << endl;
    }
    exercise_count++;
    return;
}
```c
double duration;
unsigned long int_val;
unsigned long hours;
unsigned long minutes;
unsigned long seconds;

//-------------- Code begins ------------------//
// Get the current time, and calculate session
// duration since start time. Format and
// print duration.

session_current_time = time(&tloc);
duration = session_current_time - session_start_time;

if (duration >= MAX_SESSION_DURATION) // 1 day
    { cirq << "ERROR - time out of range" << endl;

else
    { int_val = (unsigned long) duration;
        hours = int_val/NUM SECONDS_PER_HOUR;
        minutes = int_val/NUM SECONDS_PER_MINUTE;
        seconds = int_val%NUM SECONDS_PER_MINUTE;
        cout << ""This session has been active for ";
        if (hours < 1)
            { cout << minutes << " minutes, " << seconds << " seconds." << endl;

    } else
            { cout << hours << " hours, " << minutes << " minutes." << endl;

        } // Summarize other information.

    } //-------------- Code ends ------------------//

cout << "This Number of different phrases asked to translate - " << exercise_count;
cout << ""This Number of correct translations - " << correct_count;
cout << ""This Number of total responses - " << attempt_count;

} // Member function print_to_file()

//------------- Statistics -------------

#define DEBUG

cerr << "ENTRY Statistics::print_to_file*" << end;
#endif

// Automatics objects

// Automatics other

//------------- Statistics -------------

void Statistics::print_to_fileofstream * stats_stream)

#ifdef DEBUG
    cirq << "ENTRY Statistics::print_to_file*" << end;
#endif

// Automatics objects

// Automatics other

//------------- Statistics -------------

void Statistics::read(ifstream * stats_stream)

#ifdef DEBUG
    cirq << "ENTRY Statistics::read*" << end;
#endif

// Automatics objects

// Automatics other
```
#include <iostream>
#include <time.h>

using namespace std;

class Statistics {
public:
    void display();
private:
    int exercise_count;
    int attempt_count;
    int correct_count;
    int prev_duration;
};

void Statistics::display() {
    if (exercise_count > 0) {
        cout << "This session lasted:
        if (hours < 1) {
            cout << minutes << " minutes, " << seconds << " seconds."
        } else {
            cout << hours << " hours, " << minutes << " minutes."
        }
        // Summarize other information.
        //
        cout << Number of different phrases asked to translate - " << exercise_count
        cout << Number of correct translations - " << correct_count
        cout << Number of total responses - " << attempt_count
    }
    // Member function get_exercise_count().
    //
    // Member function get_attempt_count().
    //
    // Member function get_correct_count().
    //
    // Member function get_prev_duration().
    //
};

int Statistics::get_exercise_count() {
    //
    // Member function get_attempt_count().
    //
    // Member function get_correct_count().
    //
    // Member function get_prev_duration().
    //
};

int main() {
    Statistics stats;
    for (int i = 0; i < 10; i++) {
        stats.prev_duration += i;
    }
    cout << stats.prev_duration << endl;
    return 0;
}
//-------- Included files -----------
#include <iostream.h>
#include <stdlib.h>
#include "stdio.h"
#include "log.h"
#include "statistics.h"
#include "userlist.h"

//-------- External Data -----------
extern Log log;
extern Statistics stats;
extern time_t session_start_time;
extern UserList users;

//-------- Calling Syntax --------
void sys_exit_tutor()
{
    //-------- Automatics - objects --------
    //-------- Automatics - other --------
    //time_t session_end_time;
    //time_t tloc;
    //double duration;
    //unsigned long int_val;
    //unsigned long hours;
    //unsigned long minutes;
    //unsigned long seconds;

    //-------- Code begins --------
    #ifdef DEBUG
    cerr << "\t\t\tENTRY exit_tutor()" << endl;
    #endif
    // Say goodbye...
    cout << endl;
    cout << "\tYour session with the German Language Tutor is complete.\n";
    cout << endl;
}
# File: translationtree.C

This file implements the TranslationTree class.

The theory and design of this system have been documented in the accompanying thesis report.

## Included files

- `iostream.h`
- `translationtree.h`
- `patternnode.h`
- `statistics.h`
- `word_and_phrase_defs.h`

## External data

- `Statistics stats;`

## Constructor

```c
TranslationTree::TranslationTree(BasePhrase *in_bpt) { 
  BasePhrase *in_root, 
  PatternNode *in_root, 
  PhraseDescriptor *in_fixed_desc)

  if (DEBUG) 
    cerr << "\t\t\tENTRY TranslationTree CONSTRUCTOR" << endl;

  in_phrase_translator = in_bpt; 
  root_pattern = in_root; 
  fixed_phrase_desc = in_fixed_desc;
}
```

## Destructor

```c
TranslationTree::~TranslationTree() {
  if (DEBUG) 
    cerr << "\t\t\tENTRY TranslationTree DESTRUCTOR" << endl;
}
```

## Check response

```c
bool TranslationTree::check_response(BasePhrase in_phrase)
{
  if (DEBUG)
    cerr << "\t\t\tENTRY TranslationTree::check_response" << endl;
```

May 9 1993 19:12:33 models/src/translationtree.C Page 2
if (user_wants_to_try_again == FALSE)
{
    // If the user chooses not to try again after
    // diagnosis, print the expected response.
    cout << endl;
    cout << "\The system was expecting:" << endl << endl;
    cout << "\" << endl;
    expected_response.print();
    cout << endl << endl;
}
return(user_wants_to_try_again);
```c
#define DEBUG

User::User()
{
    if (DEBUG)
        cerr << "ENTRY User::User()" << endl;
    
    type = NO_USERTYPE;
    last_known_level = 0;
    login_name = new char[MAX_LOGIN_NAME_STRING];
    full_name = new char[MAX_FULL_NAME_STRING];
}

User::~User()
{
    if (DEBUG)
        cerr << "ENTRY User::~User()" << endl;

    delete [] login_name;
    delete [] full_name;
}

User::match_user(char * in_string)
```
```c
#define DEBUG 

cerr << "\DEBUG user type read is " << (int) type << endl;
#endif

cerr << "\DEBUG last knowledge level read is " << last_known_level << endl;
#endif

#if DEBUG

cerr << "\DEBUG login name read is " << login_name << endl;
#endif

cerr << "\DEBUG full name read is " << full_name << endl;
#endif

if (last_known_level != in_level) 
{
#if DEBUG

cerr << "\DEBUG current level * << last_known_level << " << in_level << endl;
#endif

cerr << "\DEBUG returning TRUE" << endl;
#endif

last_known_level = in_level;
change_made = TRUE;
}

else

#if DEBUG

cerr << "\DEBUG current level * << last_known_level << " << in_level << endl;
#endif

"cerr << "\DEBUG returning FALSE" << endl;
```
```c
67  {  
68     ifdef DEBUG  
69     cerr << "\t\t\t\tENTRY UserInterface::wrong_answer" << endl;  
69 }  
70     endif  
71     //------- Automatics - objects -------  
72     //------- Automatics - other -------  
73     bool not_satisfied = TRUE;  
74     ControlCommand choice;  
75     int int_choice;  
76     char in_char;  
77     //------- Code begins -------  
78     //------- Print message indicating that the expected -------  
79     //------- response was not input. -------  
80     cout << endl;  
81     cout << "\tThat is not the expected translation. Enter the number" << endl;  
82     cout << "\tindicating the action you would like to take:" << endl;  
83     cout << endl;  
84     //------- Print menu of options and read the user's -------  
85     //------- choice. -------  
86     while (not_satisfied)  
87     {  
88         cout << "\t" << CC_try_again << " - Try again." << endl;  
89         cout << "\t" << CC_hint << " - Have the system give you a hint." << endl;  
90         cout << "\t" << CC_correct_answer << " - Have the system give you its an-  
91         swer." << endl;  
92         cout << "\t" << CC_quit << " - Quit this exercise and go on to next." << endl;  
93         cout << "\t" << CC_exit << " - Exit the tutoring system." << endl;  
94         cout << "\t" << CC_help << " - Help." << endl;  
95         cout << endl;  
96         cout << "Choice? ";  
97         cin.get(int_char);  
98         if (int_char == '/')  
99         {  
100             choice = CC_help;  
101             }  
102             else  
103             {  
104                 cin.putback(int_char);  
105                 cin >> int_choice;  
106                 choice = (ControlCommand) int_choice;  
107             }  
108             break;  
109         }  
110         cout << "Wrong answer! Enter again" << endl;  
111         if (not_satisfied)  
112             continue;  
113         break;  
114     }  
115     endif  
116 }  
117 controlCommand UserInterface::wrong_answer()  
118 ```
switch(choice)
{
  case CC_try_again:
    cout << endl;  
    cout << "This is a menu of the options you have at this point." << endl;  
    cout << "Type the number of the option which you wish to take." << endl;  
    cout << "Your choices at this point are to try answering the exercise" << endl;  
    cout << "again, have the system give you a hint about what was wrong," << endl;  
    cout << "have the system give you the correct answer, go on to a new" << endl;  
    cout << "exercise, exit the system altogether, or view this help message." << endl;  
    cout << "not satisfied = TRUE;" << endl;  
    cout << "break;" << endl;  
    break;  
  case CC_NONE:
    default:
      cout << endl;  
      cout << "Invalid choice, please choose again:" << endl;  
      not_satisfied = FALSE;  
      break;  
    }
    return(choice);  
  }
}

ControlCommand UserInterface::after_hint(Hint * in_hint)
{
  #ifdef DEBUG
    cerr << "ENTRY UserInterface::after_hint*" << endl;  
  #endif
  // Automatics - objects
  // Automatics - other
  boolean not_satisfied = TRUE;  
  ControlCommand choice;  
  int int_choice;  
  char in_char;  

  // Code begins  

  // Request that the input Hint print itself.  

  in_hint->print();  

  // Print menu of options and read the user's choice.  

  while (not_satisfied)
  {
    cout << "t" << CC_try_again << " - Try again," << endl;  
    cout << "t" << CC_hint << " - Have the system give you another hint." << endl;  
    cout << "t" << CC_correct_answer << " - Have the system give you its answer." << endl;  
    cout << "t" << CC_quit << " - Quit this exercise and go on to next." << endl;  
    cout << "t" << CC_exit << " - Exit the tutoring system." << endl;  
    cout << "t" << CC_help << " - Help." << endl;  
    cout << endl;  
    cout << "Choice?";  
    cin.get(in_char);  
    if (in_char == 't')
    {
      choice = CC_help;  
    }
    else
    {
      cin.putback(in_char);  
      cin >> int_choice;  
      choice = (ControlCommand) int_choice;  
    }
  }
  ifdef DEBUG
    cerr << "EXIT UserInterface::after_hint*" << endl;  
  #endif

  // Parse choices - some are handled right here, others are returned to the caller.  

  switch(choice)
  {
    case CC_try_again:
      case CC_hint:
        case CC_correct_answer:
        not_satisfied = FALSE;  
        break;  
    case CC_quit:
      choice = CC_correct_answer;  
      not_satisfied = FALSE;  
    case CC_exit:
      not_satisfied = FALSE;  
      break;  
    case CC_help:
      break;  
    default:
      cout << "Invalid choice, please choose again:" << endl;  
      not_satisfied = FALSE;  
      break;  
    }
break;
case CC_exit:
    not_satisfied = FALSE;
exit_tutor();
break;
case CC_help:
    cout << endl;
    cout << "This is a menu of the options that you have at this point." << endl;
    cout << "Type the number of the option which you wish to take. Exercise" << endl;
    cout << "Your choices at this point are to try answering the exercise" << endl;
    cout << "Tag, have the system give you another hint about what was wrong." << endl;
    cout << "Have the system give you the correct answer, go on to a new session." << endl;
    cout << "Exercise, exit the system altogether, or view this help message." << endl;
    not_satisfied = TRUE;
    break;
case CC_NONE:
case CC_admin_write:
case CC_admin_delete:
case CC_admin_save:
case CC_admin_diag_fail:
case CC_admin_report_diag_fail:
case CC_admin_stats:
case CC_admin_report_stats:
case CC_admin_cum_stats:
case CC_admin_report_cum_stats:
case CC_admin_find_next:
default:
    cout << endl;
    cout << "Invalid choice, please choose again." << endl;
    not_satisfied = FALSE;
break;
} return(choice);

---

// Member function cant_diagnose().
//
ControlCommand UserInterface::cant_diagnose()
{
    //
    #ifdef DEBUG
    cerr << "ENTRY UserInterface::cant_diagnose" << endl;
    #endif

    --------- Automatics - objects ---------
    --------- Automatics - other ---------
    boolean not_satisfied = TRUE;
    ControlCommand choice;
    int int_choice;
    char in_char;
    --------- Code begins ---------

} //

// Print message indicating that the system
// could not match the expected response at all.
//
cout << endl;
cout << "That response is incorrect, but the system is unable to isolate the specific problem." << endl;
cout << "Have the system give you its answer." << endl;
cout << "Quit this exercise and go on to next." << endl;
cout << "Try again." << endl;
cout << "EXIT the tutoring system." << endl;
cout << "Help." << endl;
cout << "Choice? ";
cin.get(int_cha);
if (int_cha == '1')
    choice = CC_help;
else
    {
        cin.putback(int_cha);
        cin >> int_choice;
        choice = (ControlCommand) int_choice;
    }
#endif DEBUG

cerr << "ENTRY UserInterface::cant_diagnose" << endl;

--------- Automatics - objects ---------
--------- Automatics - other ---------
    boolean not_satisfied = FALSE;
    break;
    case CC_quit:
        choice = CC_correct_answer;
        not_satisfied = FALSE;
        break;
    case CC_help:
        cout << endl;
    } switch(choice)
cout << "This is a menu of the options that you have at this point."
  << endl;
}

void exercise(int cc_choice)
{
    switch(cc_choice)
    {
    case CC_NONE:
        case CC_admin_write:
        case CC_admin_delete:
        case CC_admin_save:
        case CC_admin_report_diag_fail:
        case CC_admin_report_stats:
        case CC_admin_report_cum_stats:
        case CC_admin_report_full:
        case CC_admin_find_next:
        default:
            cout << "Invalid choice, please choose again:"
            << endl;
            not_satisfied = TRUE;
            break;
            
    // Member function cant_diagnose_further()
    // ---------------------------------------
    
    ControlCommand(UserInterface::cant_diagnose_further());
    
    #ifdef DEBUG
    cerr << "ENTRY UserInterface::cant_diagnose_further" << endl;
    #endif

    //---------- Automatics - objects ----------
    --
    //---------- Automatics - other ----------
    --
    
    boolean not_satisfied = TRUE;
    ControlCommand(choice);
    int int_choice;
    char in_char;

    //---------- Code begins -----------------
    //
    // Print message indicating that the system
    // cannot isolate the problem any further
    //
    // than it already has.
    //
    
    
    cout << "The system cannot diagnose this any further."
    << endl;
    
    // Print menu of options and read the user's
    // choice.
    //
    
    while (not_satisfied)
    {
        cout << "t" << CC_try_again << " - Try again."
            << endl;
        cout << "t" << CC_correct_answer << " - Have the system give you its answer.
            << endl;
        cout << "t" << CC_quit << " - Quit this exercise and go on to next."
            << endl;
        cout << "t" << CC_exit << " - Exit the tutoring system."
            << endl;
        cout << "t" << CC_help << " - Help."
            << endl;
        cout << "t";
        
        cout << "Choice?";
        
        if (in_char == 't')
        {
            choice = CC_help;
        }
        else
        {
            cin.putback(in_char);
            int >> int_choice;
            choice = (ControlCommand) int_choice;
        }
        
        ifdef DEBUG
        cerr << "ENTRY UserInterface::cant_diagnose_further" << endl;
        #endif

        //---------- Automatics - objects ----------
        --
        //---------- Automatics - other ----------
        --

    switch(choice)
    {
    case CC_try_again:
    case CC_correct_answer:
        not_satisfied = FALSE;
        break;
    case CC_quit:
        choice = CC_correct_answer;
        not_satisfied = FALSE;
        break;
    case CC_exit:
        not_satisfied = FALSE;
        exit tutor();
        break;
    case CC_help:
        cout << endl;
        cout << "This is a menu of the options that you have at this point."
        << endl;
        cout << "Type the number of the option which you wish to take."
        << endl;
        cout << "Your choices at this point are to try answering the e
xercesc" << endl;
500 cout << "I'm not satisfied = " << not_satisfied << endl; 
501 break; 
502 case CC_NONE: 
503 case CC_hint: 
504 case CC_admin_write: 
505 case CC_admin_delete: 
506 case CC_admin_save: 
507 case CC_admin_news: 
508 case CC_admin_options: 
509 case CC_admin_stats: 
510 case CC_admin_report: 
511 case CC_admin_find: 
512 case CC_admin_report_stats: 
513 case CC_admin_report_summary: 
514 case CC_admin_report: 
515 case CC_admin_find: 
516 default: 
517 cout << endl; 
518 cout << "Invalid choice, please choose again: " << endl; 
519 not_satisfied = FALSE; 
520 break; 
521 return(choice); 
522 } 
523 
524 } 
525 ) ; 
526 #ifdef DEBUG 
527 #endif 
528 EOF 
529 EOF 
530 EOF 
531 EOF 
532 EOF 
533 EOF 
534 EOF 
535 EOF 
536 EOF 
537 EOF 
538 EOF 
539 EOF 
540 EOF 
541 EOF 
542 EOF 
543 EOF 
544 EOF 
545 EOF 
546 EOF 
547 EOF 
548 EOF 
549 EOF 
550 EOF 
551 EOF 
552 EOF 
553 EOF 
554 EOF 
555 EOF 
556 EOF 
557 EOF 
558 EOF 
559 EOF 
560 EOF 
561 EOF 
562 EOF 
563 EOF 
564 EOF 
565 EOF 
566 EOF 
567 EOF 
568 EOF 
569 EOF 
570 EOF 
571 EOF 
572 EOF 
573 EOF 
574 EOF 
575 EOF 
576 EOF 
577 EOF 
578 EOF 
579 EOF 
580 EOF 
581 EOF 
582 EOF 
583 EOF 
584 EOF 
585 EOF 
586 EOF 
587 EOF 
588 EOF 
589 EOF 
590 EOF 
591 EOF 
592 EOF 
593 EOF 
594 EOF 
595 EOF 
596 EOF 
597 EOF 
598 EOF 
599 EOF 
600 EOF 
601 EOF 
602 EOF 
603 EOF 
604 EOF 
605 EOF 
606 EOF 
607 EOF 
608 EOF 
609 EOF 
610 EOF 
611 EOF 
612 EOF 
613 EOF 
614 EOF 
615 EOF 
616 EOF 
617 EOF 
618 EOF 
619 EOF 
620 EOF 
621 EOF 
622 EOF 
623 EOF
cout << "Invalid choice, please choose again:" << endl;
not_satisfied = FALSE;
break;
}

// Member function sysadmin().
-----------------------------

ControlCommand UserInterface::sysadmin()
{
#ifdef DEBUG
cerr << "\t\tENTRY UserInterface::sysadmin" << endl;
#endif

// Automatics - objects ---------------
-----------------------------

// Automatics - other ---------------
--------------

while (!not_satisfied)
{

cout << "\t\tCC_exit " << - Exit the tutoring system." << endl;
cout << "\t\tCC_help " << - Help." << endl;
cout << "\t\tCC_admin_write " << - Write current log entries to file " << endl;
cout << "\t\tCC_admin_delete " << - Delete log file." << endl;
cout << "\t\tCC_admin_save " << - Save log file to new name." << endl;
cout << "\t\tCC_admin_diag_fail " << - View system diagnosis failures " << endl;
cout << "\t\tCC_admin_report_diag_fail " << - Generate report, system diagnosis failures." << endl;
cout << "\t\tCC_admin_stats " << - View individual past session statistics." << endl;
cout << "\t\tCC_admin_report_stats " << - Generate report, individual past session statistics." << endl;
cout << "\t\tCC_admin_cum_stats " << - View cumulative past session statistics." << endl;
cout << "\t\tCC_admin_report_cum_stats " << - Generate report, cumulative past session statistics." << endl;
cout << "\t\tCC_admin_report_full " << - Generate report, all information available." << endl;
cout << "Choice? ";
}
```c
35

```
cout << "\t" << CC_exit << " - Exit the tutoring system." << end;}
871 cout << "\t" << CC_help << " - Help." << end;}
872 cout << "\t" << CC_quit << " - Return to main system administrator menu
873 " << end;}
874 cout << end;}
875 cout << "Choice? ";
876 cin.get(in_char);
877 if (in_char == '?' )
878 { choice = CC_help;
879 }
880 else
881 {
882 cin.putback(in_char);
883 cin >> int_choice;
884 choice = (ControlCommand) int_choice;
885 }
886 }
887 if ifdef DEBUG
888 cerr << "\t\tDEBUG int choice is " << int_choice << end;
889 cerr << "\t\tDEBUG ControlCommand choice is " << choice << end;
890 endif
891 //--------------------------------------------//
892 // Parse choices - some are handled right here, //
893 // others are returned to the caller. //
894 //--------------------------------------------//
895 if (choice == CC_quit)
896 { not_satisfied = FALSE;
897 }
898 else if (choice == CC_exit)
899 { not_satisfied = FALSE;
900 sys_exit_tutor();
901 }
902 else if (choice == CC_help)
903 {
904 cout << end;
905 cout << "\t\tThis is a menu of the options that you have at this point
906 " << end;}
907 cout << "\t\tPlease type the number of the option which you wish to take." << end;
908 cout << "\t\tYour choices at this point are to exit the system," << end;
909 cout << "\t\tor return to the main system administrator menu." << end;
910 }
911 cout << end;
912 not_satisfied = TRUE;
913 else
914 cout << end;
915 cout << "\tInvalid choice, please choose again:" << end;
916 not_satisfied = FALSE;
917 }
918 return(choice);
stillTrying = TRUE;
cout << endl;
cout << "In order the use this system, you must have an author-
ized" << endl;
cout << "username. If you have not received a username yet, a
username" << endl;
cout << "your system administrator. If you do have a valid us-
ename,*" << endl;
cout << "type it in now to gain access to the tutoring system.
*" << endl;  
cout << endl;
}
else if (attempts >= 2)  
{  

//This is the third bad try, exit the program.  

stillTrying = FALSE;
cout << endl;
cout << "Discontinuing attempted login after three tries." << endl;
cout << "Please verify that you are using a correct username,*
<< endl;
cout << "tor see the system administrator." << endl;
cout << endl;
exit(0);  
}
else  
{

//If this is a bad attempt, but the above
//limit has not been reached, ask the user
//to try again.

attempt++;
if (stillTrying == TRUE)  
cout << endl;
cout << "That is not a correct username, please verify that* <
<< endl;
cout << "you typed what you thought you did, and that you have
not*" << endl;
cout << "used upper or lower case when not expected." << endl;  
cout << endl;
}
else  
{

//If valid username has been entered, log

//the current user, query current knowledge
//level, and set if different from previously
//known level.

stillTrying = FALSE;
current_user = search_index;
log.set_current_user((Users[current_user])->get_string());  
int in_level;
int cur_level;
if (type == STUDENT)  
{  
cur_level = users[current_user]->get_current_level();
cout << endl;
}
else  
{  
Prompt the user for his/her current
//knowledge level. Make sure it is valid.


boolean level_needed = TRUE;
while (level_needed)  
{  
cout << "Enter highest chapter completed (currently
cur_level = " << cur_level << ":") << endl;
if (in_level < MIN_KNOW_LEVEL)  
cout << "Chapter must be greater than or** equal to the minimum level," <<
MIN_KNOW_LEVEL << endl;  
else if (in_level > MAX_KNOW_LEVEL)  
cout << "The maximum chapter supported by the system is ",
MAX_KNOW_LEVEL << endl;
cout << "Please enter that number if you wish to** select the maximum" << endl;
}
else  
{  
level_needed = FALSE;  
}
if (users[current_user]->set_current_level(in_level) == TRUE)  
{  
listModified = TRUE;  
cout << endl;
cout << "Exercises will be limited to chapter * and earlier." << endl;
}
return(type);  
}  
//Member function read
//
void UserList::read()
{
    ifstream user_stream;
    // Open the user data file for input. Loop
    // and ask user objects to read data.
    //--------------------------------------------------------
    user_stream.open("users.data", ios::in);
    if (!user_stream)
    {
        cerr << "ERROR - could not open user data file." << endl;
    }
    ifdef DEBUG
    {
        cerr << "\t\t\tDEBUG users.data successfully opened." << endl;
    }
    endif

do
    users[size] = new User;
    size++;
    while (users[size-1]->read(&user_stream));
    user_stream.close();
    // Member function get_current_level().
    //--------------------------------------------------------
    int UserList::get_current_level()
    {
        ifdef DEBUG
        {
            cerr << "\t\t\tENTRY UserList::get_current_level" << endl;
        }
        endif

        return(users[current_user]->get_current_level());
    }
    // Member function write().
    //--------------------------------------------------------
    void UserList::write()
    {
        ofstream user_stream;
        // If user data has not changed, do nothing.
        // Otherwise, open the user data file for
        // output and loop through the user list
        // requesting that each write itself out to the
        // file.
        //--------------------------------------------------------
    }
}
```c
#include <iostream.h>
#include "worddescriptor.h"

int zero_based_input;
int variation;
int one_based_output;

#include "utility_gender1_var.C"

int main() {
    // Attribute.
    // The enumerated value G_none is one more than
    // the highest attribute value, so (G_none - 1)
    // is the number of possible values.  Calculate
    // the variation, modulo this number of
    // possible values.
    //
    // Return the resulting number as a Gender
    }
// Project: German Language Tutor, RIT Master's Thesis
// Purpose: This function takes a Gender attribute, and returns the
// Gender attribute which is referred to as the "second gender
// variation". This is merely the Gender attribute
// (numerically) plus 2, modulo the valid number of choices. It
// relies on the Gender enumeration.
// Notes: The theory and design of this system have been documented
// in the accompanying thesis report.
// Revisions By Reason
// 26-Apr-93 Ken Staffan v1.0 release with final thesis report.

#include <iostream.h>
#include "worddescriptor.h"

Gender utility_gender2_var(Gender in_gender)
{
    // Automatics - objects
    int zero_based_input;
    int variation;
    int one_based_output;

    // Automatics - other
    #ifdef DEBUG
    cerr << "t\t\t\tENTRY utility_gender2_var()" << endl;
    #endif
    zero_based_input = ((int) in_gender) - 1;
    // Convert input gender to a zero-based
    // integer value for math.
    zero_based_input = zero_based_input + 2;
    // The enumerated value G_none is one more than
    // the highest attribute value, so (G_none-1)
    // is the number of possible values. Calculate
    // the variation, modulo this number of
    // possible values.
    variation = (zero_based_input + 2) % (G_none-1);
    // Return the resulting number as a Gender
    return (variation);
}
May 9 1993  

---

filename: utility_not_capitalized.C
Project: German Language Tutor, RIT Master's Thesis
Purpose: This function takes a Word, and returns the same word with the first letter lower-case. Note that it assumes that the first letter is already upper-case. This is the responsibility of the caller.
Notes: The theory and design of this system have been documented in the accompanying thesis report.
Revisions By Reason
26-Apr-93 Ken Staffan v1.0 release with final thesis report.

---

#include <iostream.h>
#include <string.h>
#include "word.h"
#include "PC_defs.h"

---

Word utility_not_capitalized(Word in_word)
{
    ---
    // Obtain the current word string, and convert the first character to lower case.
    // The string is then set to the new word, and returned to the caller.
    strcpy(new_word_string, in_word.get_string(), MAX_WORD_LENGTH);
    new_word_string[0] = new_word_string[0] + CASE_CONVERSION_OFFSET;
    new_word.set_string(new_word_string); return(new_word);
}
---

### File: word.C

#### Purpose
This file implements the Word class.

#### Notes
The get_string() member function returns a pointer to the internal string. This should probably be improved, but for the time being, it is important for the caller to realize that the pointer can only be used while this object is in scope.

#### Revisions
- **09-May-93** Ken Staffan v1.0 release with final thesis report.

---

```c
#include <iostream.h>
#include "word.h"

Word::Word()
{
    #ifdef DEBUG
        cerr << "\t\t\t\tENTRY Word CONSTRUCTOR 1" << endl;
    #endif

    // Constructor 1 of 3.
    //
    // Constructor 2 of 3.
    //
    // Constructor 3 of 3.
    //
    index = 0;
    word = new char[MAX_WORD_LENGTH];
    type = WD_UNKNOWN_TYPE;
    count = C_none;
    gender = G_none;
    language = L_none;
};

Word::Word(char * in_static, WD_Type in_type, Count in_count, Gender in_gender,
          Language in_language)
{
    #ifdef DEBUG
        cerr << "\t\t\t\tENTRY Word CONSTRUCTOR 2" << endl;
    #endif

    cerr << "\t\t\t\tENTRY (for knowledge base)" << endl;
    #endif
}

---

### Set internal data based on input.
---

```
```c
while (!end_of_word && cin.get(ch))
{
    if (cin_word != this)
    {
        delete [] word;
        word = new char[strlen(in_word.word)+1];
        strcpy(word, in_word.word);
        index = in_word.index;
        type = in_word.type;
        count = in_word.count;
        gender = in_word.gender;
        language = in_word.language;
        return *this;
    }

    // Member function set_string()---------------------------
    // Copy the contents of the input string to the
    // internal string.
    // -----------------------------------------------------
    strcpy(word, in_string, MAX_WORD_LENGTH);
}

// Member function read()-----------------------------------

boolean Word::read()
{
    // Automatics - objects
    // Automatics - other
    char ch;
    int ch_index = 0;
    boolean end_of_line = FALSE;
    boolean end_of_word = FALSE;

    // Code begins
    // Read characters from the input stream
    // until end-of-word is encountered.
```
void Word::print()
{
    if (DEBUG)
    {
        cerr << "\t\t\tENTRY Word::print" << endl;
    }
    #ifndef DEBUG
    #endif
    //---------- Automatics - objects ----------
    //---------- Automatics - other ----------
    //---------- Code begins ----------
}

void Word::set_index(int in_index)
{
#ifndef DEBUG
    cerr << "\t\t\tENTRY Word::set_index" << endl;
#else
    cout << "\t\t\tENTRY Word::set_index" << endl;
#endif
    //---------- Automatics - objects ----------
    //---------- Automatics - other ----------
    //---------- Code begins ----------
    index = in_index;
}

void Word::set_language(Language in_lang)
{
    if (DEBUG)
    {
        cerr << "\t\t\tENTRY Word::set_language" << endl;
    }
    //---------- Automatics - objects ----------
    //---------- Automatics - other ----------
    //---------- Code begins ----------
    language = in_lang;
}

void Word::set_type(WD_Type in_type)
{
#ifndef DEBUG
    cerr << "\t\t\tENTRY Word::set_type" << endl;
#else
    cout << "\t\t\tENTRY Word::set_type" << endl;
#endif
    //---------- Automatics - objects ----------
    //---------- Automatics - other ----------
    //---------- Code begins ----------
}
void Word::set_count(int in_count)
{
    cout << "t\t\t\tENTRY Word::set_count" << endl;
    count = in_count;
}

Language Word::get_language()
{
    cout << "t\t\t\tENTRY Word::get_language" << endl;
    return(language);
}

int Word::get_index()
{
    cout << "t\t\t\tENTRY Word::get_index" << endl;
    return(index);
}

WD_Type Word::get_type()
{
    cout << "t\t\t\tENTRY Word::get_type" << endl;
    return(type);
}

Gender Word::get_gender()
{
    cout << "t\t\t\tENTRY Word::get_gender" << endl;
    return(gender);
}

Count Word::get_count()
{
    cout << "t\t\t\tENTRY Word::get_count" << endl;
    return(count);
}
void WordDescriptor::set_gender(Gender in_gender)
{
    ifdef DEBUG
    cerr << "\t\t\tENTRY WordDescriptor::set_gender" << endl;
    endif

    // Constructor 1.

    type = in_type;
    count = in_count;
    gender = in_gender;
    language = in_lang;

    // Constructor 2.

    // Destructor.

    WordDescriptor::~WordDescriptor();

    #ifdef DEBUG

    void WordDescriptor::set_language(Language in_language)
{
    ifdef DEBUG
    cerr << "\t\t\tENTRY WordDescriptor::set_language" << endl;
    endif

    // Constructor.

    // Destructor.

    #ifndef DEBUG

    WordDescriptor::WordDescriptor()
{

    #ifdef DEBUG

    // Destructor.

    WordDescriptor::~WordDescriptor();

    #endif
Count WordDescriptor::get_count()

```c
#ifdef DEBUG
    cerr << "\t\tENTRY WordDescriptor::get_count" << endl;
#endif

//-------- Automatics - objects ---------------------
//-------- Automatics - other ---------------------
//-------- Code begins ---------------------

    return(count);
```

`Gender WordDescriptor::get_gender()`

```c
# ifdef DEBUG
    cerr << "\t\tENTRY WordDescriptor::get_gender" << endl;
#endif

//-------- Automatics - objects ---------------------
//-------- Automatics - other ---------------------
//-------- Code begins ---------------------

    return(gender);
```

`Language WordDescriptor::get_language()`

```c
# ifdef DEBUG
    cerr << "\t\tENTRY WordDescriptor::get_language" << endl;
#endif

//-------- Automatics - objects ---------------------
//-------- Automatics - other ---------------------
//-------- Code begins ---------------------

    return(language);
```

`boolean WordDescriptor::no_dynamics()`

```c
# ifdef DEBUG
    cerr << "\t\tENTRY WordDescriptor::no_dynamics" << endl;
#endif
```
```c
 gender = in_dyna.get_gender();
#endif DEBUG
  cerr << "	DEBUG resolved dynamic attribute gender for WD" << endl;
#endif
    if (language == L_DYNAMIC)
      {  
        #ifdef DEBUG
          cerr << "	DEBUG resolving dynamic attribute lang for WD" << endl;
        #endif
          language = in_dyna.get_language();
      }
```

5-May-93, Kenneth E. Staffan

When saved to floppy disks, the German Language Tutor system was left
in a "stable state". All source files, scripts, and build procedures were
baseline in SCCS. The system was then built, leaving executables and
reference copies of all files (source code in /src directories, include
files in /hdr directories, build procedures and scripts in the top-level and
component-level directories). One file, compound. C compile with
warnings. These are due to the use of the 'regex' utility, and the
debug_print() member function. I didn't bother too much with them, because
they do not affect the system at run-time, debug_print() is only debug
code, and I wanted to switch to another regular expression package anyway.

The system was executed, leaving behind debug and system administration
logs. The entire directory structure was then put on floppies.

Anyone interested in using, enhancing, or otherwise digging into this
code, should reference the corresponding thesis report, which also
contains user and system administrator manuals.

The following summarizes the directories and files saved:

tutor - top-level directory

    READ_ME - this file.
    /SCCS - sccs controlled top-level files.
    baseline_all
    checkin_all
    checkout_all
    print_all
    split - supporting scripts (all in SCCS also).
    /hdr - top-level include files (all in SCCS also).
    makefile - system build procedure (in SCCS also).
    users.data - system input - valid usernames.
    git - system executable (run using split script).
    debug.log - system output - verbose debugging output.
    git.log - system output - run-time gathered usage data.

/construction
/dictionary
/log
/models
/session
/users - component-level directories. All components are
similar, so the 'users' component is broken down
further to illustrate:

    /SCCS - sccs controlled component-level files.
    /hdr - component-level include files (all in SCCS also).
    /lis - compiler-generated listing files (not currently used).
    /obj - compiler-generated individual object modules.
    /src - component-level source files (all in SCCS also).
    makefile - component build procedure (in SCCS also).
    users.o - component object module, to be linked at top-level.

NOTE: Because the *all* scripts do some pretty global things, they should
be used with care. I always made a backup copy of the directory
structure before I ran them - just in case!
The procedure for building the German language tutor thesis program (gli).

This makefile may invoke makefiles at the sub-system level, located below
this directory.

Modification History:

<table>
<thead>
<tr>
<th>When</th>
<th>Who</th>
<th>Why</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07Nov92</td>
<td>Ken Staffan</td>
<td>Initial sub version.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28Nov92</td>
<td></td>
<td>First config - single sub-system, 'models'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24Mar93</td>
<td></td>
<td>Added other sub-systems and global header files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08Apr93</td>
<td></td>
<td>Phases.h became compoundstring.h.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12Apr93</td>
<td></td>
<td>Added hint.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17Apr93</td>
<td></td>
<td>Ensure that dictionary builds before phrase models.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25Apr93</td>
<td></td>
<td>Added userinterface.defs.h, word_and_phrase.defs.h and worddescriptor.defs.h.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maintain memory of previous runs.

KEEP_STATE:

The "make" command to be passed to sub-system builds. (Passes down flags which were used when invoking this top-level makefile.)

MAKE = make $(MAKEFLAGS)

Sub-directories of necessary sub-systems. (List alphabetically)

SUB_DIRS =

construction

dictionary

log

models

session

users

The expected outputs of the above listed sub-systems. (IMPORTANT NOTE: the order in which these are listed affects the order in which certain static objects are created. These are constructed roughly from the bottom up in this list. dictionary.o is listed below models.o so that the dictionary is available before constructing basephrases, which use the dictionary to determine knowledge level.)

SUB_DIRS =

construction/construction.o

log/log.o

models/models.o

dictionary/dictionary.o

session/session.o

users/users.o

Global include files. The GBL_HDRS macro is passed down to sub-systems, so this is where any include files which may be shared across sub-systems are defined. The HDRS macro is used at this level for "building" these header files (basically just pulling them out of configuration management).

GBL_HDRS=

..hdr/basephrase.h

..hdr/compounding.h

..hdr/dictionary.h

..hdr/hint.h

..hdr/logdefs.h

..hdr/log.h

..hdr/phrasemodellist.h

..hdr/statistics.h

..hdr/statisticsdefs.h

..hdr/strtype.h

..hdr/userdefs.h

..hdr/userinterface.h

..hdr/userinterfacedefs.h

..hdr/wrod.h

..hword_and_phrase.defs.h

..hworddescriptor.h

..hworddescriptordefs.h

RDRS=

..hdr/basephrase.h

..hdr/compounding.h

..hdr/dictionary.h

..hdr/hint.h

..hdr/logdefs.h

..hdpolog.log.h

..hdr/phrasemodellist.h

..hdr/statistics.h

..hdr/statisticsdefs.h

..hdr/strtype.h

..hdr/userdefs.h

..hdr/userinterface.h

..hdr/userinterfacedefs.h

..hdr/userlist.h

..hdpoword.h

..hword_and_phrase.defs.h

..hworddescriptor.h

..hworddescriptordefs.h

The rules for 'building' global include files.

hdr/ht.h: SCCS/s. $h

@******************************************************************************

Get global include files.

cd hdr; $acc -d..; get $(DF); cd ..; 

******************************************************************************

Actual build procedure begins here.

******************************************************************************

all: gbl_incs sub_systems g1t

#
# Target for updating global header files. (No explicit action - forces the
# SCCS extraction, if needed).

gbl_incs: $[HDRS]

### Update global header files.

# Target for updating sub-systems.

sub_systems:

for i in $(SUB_DIRS); do 
  cd $i; $(MAKE) GBL_HDRS="$[GBL_HDRS]" ; cd .. ; 
  done

# Target for the actual tutoring system executable - "glit"

glit: $(SUB_PARTS)

### Build the executable - "glit"

CC $(SUB_PARTS) -o glit
The procedure for building the construction subsystem - one subsystem in the German language tutor thesis program.

Modification History:

When  Who   Why
---  ---  ----
26Mar93肯 Staffan Initial version.
04Apr93 " Renamed phrase_modifier.h to PC_defs.h, PM *.c to
PC*.c, and data PCA_phrase_modifiers.C to data PCA_defs.c
08Apr93 " Added utility_gend1_var.c, PCA_gend2_first_word.c,
utility_gend2_var.C, PC wrong third word.C,
PC wrong fourth word.C, PC wrong first and second
word.C, PC nonCap_second_word.C, utility_not-
capitalized.C.
20Apr93 " Added PC_extra_article_fourth.

Maintain memory of previous runs.

.include files which may be used by source in this subsystem. This
includes both the local include files, and the global include files
which are available to all subsystems.

#HDRS= \$(GLB_HDRS) \$
# hdr/PC_defs.h \$
# hdr/dynamicworddescriptor.h

# List of object modules which are built for this subsystem.

#OBJS= \obj/PC_extra_article_fourth.o \$
# obj/PC_gend1_first_word.o \$
# obj/PC_gend2_first_word.o \$
# obj/PC_noncap_second_word.o \$
# obj/PC_wrong_first_and_second_word.o \$
# obj/PC_wrong_first_word.o \$
# obj/PC_wrong_fourth_word.o \$
# obj/PC_wrong_second_word.o \$
# obj/PC_wrong_third_word.o \$
# obj/compoundstring.o \$
# obj/data_PC_defs.o \$
# obj/dynamicworddescriptor.o \$
# obj/phrasedescriptor.o \$
# obj/utility_gend1_var.o \$
# obj/utility_gend2_var.o \$
# obj/utility_not_capitalized.o \$
# obj/word.o \$
# obj/worddescriptor.o

# Implicit build rules for source code and include files.

src/tinyos/SCCS/a.1.C

"Get construction source code file
    cd src; sccs -d../ $(@F); cd ..;

hdr/td.h: SCCS/a.1.h

"Get construction include file
    cd hdr; sccs -d../ $(@F); cd ..;

# Actual build procedure begins here.

# Build a single construction object module
    1d -r $(@OBJS) -o construction.o

$:@OBJS: src/$$[$F::.o=.c] $:@HDRS)

"Compile construction source file
    cd src; /usr/lang/SOC1.0/CC -DDEBUG +I -I../hdr -I..../hdr $(@F:.o=.c)
    \ ; rm *.c; \$
    mv $(@F) ./obj/$(@F); cd ..

# Compile construction source files
    cd src; CC -DDEBUG +I -I../hdr -I..../hdr $(@F:.o=.c) \$
    mv $(@F:.o=.c) ../lib; \$
    mv $(@F) ./obj/$(@F); cd ..
The procedure for building the dictionary subsystem - one subsystem in the
German language tutor thesis program.

Modification History:

- 26Mar93 Ken Staffan Initial version.
- 05Apr93 * Renamed dictentry to dictionaryentry.
- 17Apr93 * Added dictionary_defs.h.

# Maintain memory of previous runs.

.KEEP_STATE:

- Include files which may be used by source in this subsystem. This
  includes both the local include files, and the global include files
  which are available to all subsystems.

HDBS= \n  $(GBL_HDBS) \n  hdr/dictionary_defs.h \n  hdr/dictionaryEntry.h

# List of object modules which are built for this subsystem.

OBJS= \n  obj/data_dictionary.o \n  obj/dictionary.o \n  obj/dictionary_entry.o

# Implicit build rules for source code and include files.

src/*.C: SCCS/s.r.C
  # Get dictionary source code file
  cd src; sccs -d.. get $(S); cd ..;

hdr/*.h: SCCS/s.r.h
  # Get dictionary include file
  cd hdr; sccs -d.. get $(S); cd ..;

# Actual build procedure begins here.

dictionary.o: $(OBJS)
  # Build a single dictionary object module
  ld -r $(OBJS) -o dictionary.o
The procedure for building the log subsystem - one subsystem in the
German language tutor thesis program.

Modification History:

When  Who    Why
-----  ---    ---
26Mar93 Ken Staffan Initial version.

# Maintain memory of previous runs.

.KEEP_STATE:

# Include files which may be used by source in this subsystem. This
# includes both the local include files, and the global include files
# which are available to all subsystems.

HDRS= \$(GBL_HDRS) \\n  hdr/logentry.h

# List of object modules which are built for this subsystem.

OBJS= \n  obj/log.o \n  obj/logentry.o

# Implicit build rules for source code and include files.

  src/t.c: SCSS/a.t.c
  #*************************** Get log source code file
  cd src; accs -d./ get $(GF); cd ..;

  hdr/t.h: SCSS/a.t.h
  #*************************** Get log include file
  cd hdr; accs -d./ get $(GF); cd ..;

# Actual build procedure begins here.

#*************************************************************************

log.o: $(OBJS)
  #*************************** Build a single log object module
  ld -r $(OBJS) -o log.o

$(OBJS): src/$($GF:.C) $(HDRS)
  #*************************************************************************

Compile log source file
May 9 1993 19:12:37 models/makefile Page 1

1 The procedure for building the models subsystem - one subsystem in the
2 German language tutor thesis program.
3
4 Modification History:
5
6 When Who Why
7 9 07Oct92 Ken Staffan Initial version.
8 29Nov92 * Added session.o, userlist.o and userlist.o.
9 30Nov92 * Added session.h, stdtype.h, modellist.h, modellist.o,
10   dictionary.o, dictionary.h, dictionary.o, dictionary.h,
11   word.o, word.h, worddescriptor.o, and worddescriptor.h.
12 05Dec92 * Added basephrase.h & .o.
13 19Dec92 * Added basephrase and basephrasepool.h and .o.
14 20Dec92 * Renamed modelist* to phrasemodiist, added
15   basephrasepool.o.
16 28Dec92 * RENAME data basephrasepool to data phrasemodiist,
17   added phrasemod.h & .c
18 30Dec92 * Added data_dictionary.c.
19 31Dec92 * Added phrasemodifier.h & .c and phrase.h & .c.
20 16Jan93 * Takes a lot of space. Should do on demand only.
21 17Jan93 * Added pattern, patterntree, phasemodifier and
22   basephrase and basephrasepool.h & .c Alphabetized lists.
23 23Jan93 * Renamed patterntree to translationtree.
24 08Feb93 * Renamed link file from .ink to .o for RIT CC.
25 09Feb93 * Don't generate ..c until RIT quota problem resolved.
26 20Feb93 * Added dynamicworddescriptor.*.
27 25Feb93 * Added PM, data phrase_modifiers.C & phrase_modifiers.h
28   Deleted phasemodifier.h & .c
29 26Feb93 * Added -DEDEBUG to CC command.
30 28Feb93 * Added PM Gendered first_word.C, user.h & .c, and
31   data users.C
32 06Mar93 * Added BTP_no_translation.C.
33 07Mar93 * Added useinterface.h & .c.
34 13Mar93 * Added log.h & .c, logentry.h & .c, exit_tutor.C.
35 15Mar93 * Added statistics.h & .c
36 24Mar93 * First cut at breaking out all other subsystems.
37 05Apr93 * Renamed pattern* to patternnode*.
38 10Apr93 * Removed obsolete basephrase and basephrasepool.h & .c
39 12Apr93 * Added hint.C
40 28Apr93 * Added BTP drop_fourth_word.C
41
42 # Maintain memory of previous runs.
43
44 .KEEP_STATE:
45
46 # Include files which may be used by source in this subsystem. This
47 # includes both the local include files, and the global include files
48 # which are available to all subsystems.
49
50 # # HDBS=
51 ($GBL_HDBS) \n52   hdr/basephrasepool.h \n53   hdr/patternnode.h \n54   hdr/phrasemod.h \n55 #
56
May 9 1993 19:12:37 models/makefile Page 2
67 #hdr/translationtree.h
68
69 # List of object modules which are built for this subsystem.
70
71 OBJS=
72 obj/BTP drop_fourth_word.o \n73 obj/BTP_no_translation.o \n74 obj/basephrase.o \n75 obj/basephrasepool.o \n76 obj/data_phrasemodiist.o \n77 obj/hint.o \n78 obj/patternmode.o \n79 obj/phrasemod.o \n80 obj/phrasemodiist.o \n81 obj/translationtree.o
82
83 # Implicit build rules for source code and include files.
84
85 src/*.c: SCSS/a.a.c
86 
87 #----------------------------- Get models source code file
88 cd src; scs -d ./ $(@F); cd ..;
89
90 hdr/*.h: SCSS/a.a.h
91 
92 #----------------------------- Get models include file
93 cd hdr; scs -d ./ $(@F); cd ..;
94
95
96 #----------------------------------------------------------
97 # Actual build procedure begins here.
98 #----------------------------------------------------------
99
100 models.o: $(OBJS)
101 
102 #----------------------------- Build a single models object module
103 1d -r $(OBJS) -o models.o
104
105
106
107 $OBJ2S: src/$F.o.o=.C ($(HDRS)
108 
109 #----------------------------- Compile models source file
110 cd src; /usr/lang/SCL0/CC -DEDEBUG +t -c -I../hdr -I../.hdr $((F.o.o=.C)
111 \n112 rm *o.c; \n113 mv $((F) ..obj/$((F); cd ..
114
115
116
117 $OBJ2S: src/$F.o.o=.C ($(HDRS)
118 
119 #----------------------------- Compile models source file
120 cd src; /usr/lang/SCL0/CC -DEDEBUG +t -c -I../hdr -I../.hdr $((F.o.o=.C)
121 \n122 rm *o.c; \n123 mv $((F.o.o=.C) ..//lis; \n124 mv $((F) ..//obj/$((F); cd ..
125
126
127
128
The procedure for building the session subsystem - one subsystem in the
German language tutor thesis program.

Modification History:

When    Who   Why
-----    ---   ---
24Mar93  Ken Staffan  Initial version.
20Apr93  *   Added statistics_defs.h
26Apr93  *   Added sys_exit_tutor();

Maintain memory of previous runs.

.KEEP_STATE:

Include files which may be used by source in this subsystem. This
includes both the local include files, and the global include files
which are available to all subsystems.

HDRS= \$(GBL_HDRS) \n    hdr/session.h

List of object modules which are built for this subsystem.

OBJS= \n    obj/exit_tutor.o \n    obj/session.o \n    obj/statistics.o \n    obj/sys_exit_tutor.o \n    obj/userInterface.o

Implicit build rules for source code and include files.

src/*.c: SCCS/s.t.c
        ************** Get session source code file
        cd src; sccs -d ./ $(@F); cd .;

hdr/*.h: SCCS/s.t.h
        ************** Get session include file
        cd hdr; sccs -d ./ $(@F); cd .;

******************************************************************************

Actual build procedure begins here.

******************************************************************************

session.o: $(OBJS)
        **************************** Build a single session object module
The procedure for building the users subsystem - one subsystem in the German language tutor thesis program.

Modification History:

- When Who Why
- --- --- ---
- 26Mar93 Ken Staffan Initial version.

- Maintain memory of previous runs.

KEEP_STATE:

/* Include files which may be used by source in this subsystem. This includes both the local include files, and the global include files which are available to all subsystems. */

HDRS= \$(GBL_HDRS) \$
    hdr/user.h

# List of object modules which are built for this subsystem.

OBJS= \$ (obj/user.o \$
    obj/userlist.o

# Implicit build rules for source code and include files.

src/*.C: SCCS/a.c
    # Get users source code file
    cd src; sccs -d ./ get $(SDF); cd ..

hdr/*.h: SCCS/a.h
    # Get users include file
    cd hdr; sccs -d ./ get $(SDF); cd ..

# Actual build procedure begins here.

# Build a single users object module

users.o: $(OBJS)
    # Build a single users object module
    ld -r $(OBJS) -o users.o
# German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993

---

# baseline_all

---

This script could be destructive, so the setup must be done manually before running this script. The procedure would be:

- Execute the checkout_all script to extract all files.
- Remove all files from the /SCCS subdirectories.
- Run this script to re-create baseline versions of all source.
- If it appears that the script executed successfully, execute the cleanup_all script to remove all *. (note leading comma) files left by SCCS, and all .h and .C in the directories which have comma-files.

Modification History:

# Revisions By Reason
--- ----
26Apr93 Ken Staffan v1.0 release with final thesis report.

# Create top-level .h files, scripts, and build procedures.

```bash
# sccs create README
sccs get README
# sccs create users.data
sccs get users.data
# sccs create *.h
sccs create makefile
sccs get makefile
chmod 700 makefile
# sccs create split
sccs get split
chmod 700 split
# sccs create baseline_all
sccs get baseline_all
chmod 700 baseline_all
# sccs create checkin_all
sccs get checkin_all
chmod 700 checkin_all
# sccs create checkout_all
sccs get checkout_all
chmod 700 checkout_all
# sccs create print_all
sccs get print_all
chmod 700 print_all
# sccs create cleanup_all
sccs get cleanup_all
chmod 700 cleanup_all
```

# Change to each component sub-directory, and create .h and .C files,
May 9 1993 19:12:37  checkin_all  Page 1

1    # German Language Tutor - RIT Master's Thesis - Kenneth E. Staffan - 1993
2    #
3    # checkin_all
4    #
5    # This script will replace all source files which were extracted for
6    # change by "checkout_all". The user will be prompted for the SCCS
7    # comment. (A future enhancement could be to have the script ask once,
8    # and feed the comment into SCCS). This script does not need to be
9    # updated when source files are added or deleted (though it will fail
10    # if new source files exist which have not yet been "sccs create"d.
11    # It assumes that all of the files are in the directory above their
12    # respective /SCCS directory.
13    #
14    # Modification History:
15    #
16    # Revisions  By     Reason
17    #--------  ----     ------
18    # 26Apr93  Ken Staffan v1.0 release with final thesis report.
19    #
20    #
21    # Replace the top-level .h files, scripts, and build procedures.
22    #
23    #
24    # sccs delta *.h
25    # sccs delta READ_ME
26    # scc get READ_ME
27    #
28    # sccs delta users.data
29    # scc get users.data
30    #
31    # sccs delta makefile
32    # scc get makefile
33    # chmod 700 makefile
34    #
35    # sccs delta split
36    # scc get split
37    # chmod 700 split
38    #
39    # sccs delta baseline_all
40    # scc get baseline_all
41    # chmod 700 baseline_all
42    #
43    # sccs delta checkin_all
44    # scc get checkin_all
45    # chmod 700 checkin_all
46    #
47    # sccs delta checkout_all
48    # scc get checkout_all
49    # chmod 700 checkout_all
50    #
51    # sccs delta print_all
52    # scc get print_all
53    # chmod 700 print_all
54    #
55    # sccs delta cleanup_all
56    # scc get cleanup_all
57    # chmod 700 cleanup_all
58    #
59    # Change to each component sub-directory, and replace .h and .C files and
60    # build procedures.

May 9 1993 19:12:37  checkin_all  Page 2

61    #
62    # cd construction
63    # sccs delta *.h
64    # sccs delta makefile
65    # chmod 700 makefile
66    # cd ../dictionary
67    # sccs delta *.h
68    # sccs delta *.C
69    # sccs get makefile
70    # chmod 700 makefile
71    # cd ../log
72    # sccs delta *.h
73    # sccs delta makefile
74    # sccs get makefile
75    # chmod 700 makefile
76    # cd ../models
77    # sccs delta makefile
78    # sccs delta*.c
79    # sccs get makefile
80    # chmod 700 makefile
81    # cd ../sessions
82    # sccs delta*.h
83    # sccs delta*.C
84    # sccs delta makefile
85    # sccs get makefile
86    # chmod 700 makefile
87    # cd ../users
88    # sccs delta*.h
89    # sccs delta*.C
90    # sccs delta makefile
91    # sccs get makefile
92    # chmod 700 makefile
93    # cd ...
94    # sccs delta*.h
95    # sccs delta*.C
96    # sccs delta makefile
97    # sccs get makefile
98    # chmod 700 makefile
99    #
This script will extract all source files (top-level .h, and component-level .c) and all build procedures and scripts, from SCCS for edit (change). There is no equivalent script to get read-only versions, since these will normally be left in the various /hdr and /src subdirectories after a build. This will leave the files in the directory above the /SCCS directory from which they were extracted. The script is intended to be run from the top-level directory, and will need to be updated if any source files are added or deleted.

NOTE that the script first deletes any versions in the directories, so care should be taken not to invoke this script while individual files are checked out (it will overwrite them).

Modification History:

Revisions By Reason
-------- -- ------
26Apr93 Ken Staffan v1.0 release with final thesis report.

Extract the top-level .h files, scripts, and build procedures.

rm -f README
rm -f users.data
rm -f makefile
rm -f split
rm -f baseline_all
rm -f checkin_all
rm -f checkout_all
rm -f print_all
rm -f cleanup_all
rm -f *.h
rm -f *.c
sccs edit README
sccs edit users.data
sccs edit makefile
chfnod 700 makefile
sccs edit split
chfnod 700 split
sccs edit baseline_all
chfnod 700 baseline_all
sccs edit checkin_all
chfnod 700 checkin_all
sccs edit checkout_all
chfnod 700 checkout_all
sccs edit print_all
chfnod 700 print_all
sccs edit cleanup_all
chfnod 700 cleanup_all
sccs edit basephrase.h
sccs edit compoundstring.h
sccs edit dictionary.h
sccs edit hint.h
sccs edit log.h
sccs edit log_defs.h

cd ..
dictionary
rm -f makefile
rm -f *.h
rm -f *.c
sccs edit makefile
sccs edit dictionary_defs.h
sccs edit dictionary_entry.h
sccs edit data_dictionary.h
sccs edit data_dictionary.C
sccs edit dictionaryentry.C
sccs edit dictionary_entry.C
cd ..
log
rm -f makefile
rm -f *.h
rm -f *.c
sccs edit makefile
sccs edit logentry.h
sccs edit logentry.C
cd ..
models
rm -f makefile
rm -f *.h
rm -f *.c
sccs edit makefile
May 9 1993 19:12:37 checkout_all Page 3

133 sccs edit basephaselist.h
134 sccs edit basephrasepool.h
135 sccs edit patternnode.h
136 sccs edit phasemodel.h
137 sccs edit translationtree.h
138 sccs edit BPT_drop_fourth_word.C
139 sccs edit BPT_no_translation.C
140 sccs edit basephrase.C
141 sccs edit basephaselist.C
142 sccs edit basephrasepool.C
143 sccs edit data_phrasemodellist.C
144 sccs edit hint.C
145 sccs edit patternnode.C
146 sccs edit phasemodel.C
147 sccs edit phrasemodellist.C
148 sccs edit translationtree.C
149 cd ..//session
150 rm -f makefile
151 rm - f *.h
152 rm - f *.C
153 sccs edit makefile
154 sccs edit session.h
155 sccs edit exit_tutor.C
156 sccs edit session.C
157 sccs edit statistics.C
158 sccs edit sys_exit_tutor.C
159 sccs edit userinterface.C
160 cd ..//users
161 rm -f makefile
162 rm - f *.h
163 rm - f *.C
164 sccs edit makefile
165 sccs edit user.h
166 sccs edit user.C
167 sccs edit userlist.C
168 cd ..
# cleanup_all

This script is only intended to be executed after a successful execution of baseline_all. It cleans up the SCSS create residual files.

Modification History:

- 26Apr93 Ken Staffan v1.0 release with final thesis report.

Remove top-level residual files.

```bash
rm -f, *
rm -f *.h
```

Remove component-level residual files.

```bash
cd construction
rm -f, *
rm -f *.h
rm -f *.C
```

```bash
cd ../dictionary
rm -f, *
rm -f *.h
```

```bash
cd ../log
rm -f, *
rm -f *.h
```

```bash
cd ../models
rm -f, *
rm -f *.h
```

```bash
cd ../session
rm -f, *
rm -f *.h
```

```bash
cd ../users
rm -f, *
rm -f *.h
```

```bash
cd ..
```
print_all

This script prints all source code, scripts and build procedures in
the desired listing order. Note that the printing command may need
change on a different machine.

Modification History:

Revisions  By Reason
----------  ------  -------
  26Apr93  Ken Staffan  v1.0 release with final thesis report.

Print all include files in alphabetical order.

a2ps

construction/src/PC_wrong_first_and_second_word.C
construction/src/PC_wrong_first_word.C
construction/src/PC_wrong_fourth_word.C
construction/src/PC_wrong_second_word.C
construction/src/PC_wrong_third_word.C
models/src/basephrase.C
models/src/basephrasealist.C
models/src/basephrasepool.C
construction/src/compoundstring.C
construction/src/data_PC_defs.C
dictionary/src/data_dictionary.C
models/src/data_phraseorderedlist.C
construction/src/dynamicworddescriptor.C
session/src/exit_tutor.C
models/src/int.C
log/src/log.C
log/src/logentry.C
models/src/patternnode.C
construction/src/phrasedescriptor.C
models/src/phraseorderedlist.C
session/src/session.C
session/src/sessionlist.C
session/src/sys_exit_tutor.C
models/src/translationtree.C
users/src/user.C
session/src/userinterface.C
users/src/userlist.C
construction/src/utility_gender1_var.C
construction/src/utility_gender2_var.C
construction/src/utility_notcasecapitalized.C
construction/src/wdword.C
construction/src/worddescriptor.C

lpr -Pagate

Print all source files in alphabetical order.

a2ps

models/src/BPT_drop_fourth_word.C
models/src/BPT_no_Translation.C
construction/src/PC_extra_article_fourth.C
construction/src/PC_gender1_first_word.C
construction/src/PC_gender2_first_word.C
construction/src/PC_noun_case_second_word.C

May 9 1993 19:12:37  print_all  Page 2

May 9 1993 19:12:37  print_all  Page 1
split

This script allows the system executable to be run, redirecting any
debug or error output to the file debug.log. (Leaving the standard
user interface displaying to the screen.)

Modification History:

Revisions  By  Reason
---------  ---  -------
26Apr93  Ken Staffan  v1.0 release with final thesis report.

sh -c "glt 2> debug.log"