2002

Designing programming courses for ET students

Anthony Trippe

Session 3547

Designing Programming Courses For ET Students

Anthony P. Trippe

Rochester Institute of Technology
Electrical, Computer and Telecommunications Engineering Technology Department

Abstract

Rochester Institute of Technology offers a three-course technical programming sequence for Engineering Technology students. All three courses are required for Computer Engineering Technology students. The first two courses are required for Telecommunications ET students. Electrical and Civil ET students are required to take only the first course of the sequence.

This paper reviews and discusses the events and experiences associated with the development and initial conduct of this sequence of C++ programming courses. It details how the courses were designed to assist ET students to better succeed in higher level courses taken later in their program sequence. Foundation elements of these courses include C++ procedural and object oriented grammar and syntax, programming structures and data structures. The paper illustrates how secondary elements of a technical programming course can be selected so as to additionally promote and encourage student learning of techniques for applied technical problem solving, technical writing, software engineering, project management, team dynamics and ethics.

Introduction

Computer Programming skills are an important recommended part of educational program curricula in many disciplines (1), (2). Engineering Technology (ET) is no exception. Computer Competency for ET students is defined by ABET criteria (3). The criteria states:

> "Engineering technicians and technologists are dependent upon the computer to effectively perform their job functions. It is therefore essential that students acquire a working knowledge of computer usage. Instruction in applications of software for solving technical problems and student practice within appropriate technical courses is required for all programs. Additionally in Baccalaureate degree programs, instruction must be included in one or more of the computer languages commonly used in the practice of engineering technology. Following formal instruction or demonstrated proficiency in computing skills, baccalaureate

students should gain experience using programming skills in technical courses to an extent appropriate for the discipline."

The ET Programming Course Sequence

Rochester Institute of Technology offers a number of Engineering Technology baccalaureate degrees in a wide range of disciplines (4). In line with ABET criteria, it was decided to develop programming courses which address the unique, specific needs of ET students. Unlike Information Technology or Computer Science students, ET students need to understand hardware and interface issues associated with programming and software development. In addition, it was planned that these courses be structured to better prepare RIT's ET students for success in courses taken later in their program sequence.

In June of 2000, the author was assigned responsibility for the development and initial teaching of a three-course C++ technical programming sequence. The courses were to be initially presented in the classroom environment but that distance learning versions would subsequently be offered.

At Rochester Institute of Technology, the three course sequence (4) is required for all Computer Engineering Technology students. The first two courses are required for Telecommunications ET students. Electrical and Civil ET students are required to take the first course of the sequence. For the Electrical, Computer and Telecommunications ET students, these courses are taken early in the program (first and second year) so that the programming concepts, knowledge and skills can be applied to solving scientific and engineering problems encountered in upper level courses.

The foundation objective for the course sequence is that students learn the grammar and syntax of the C++ language and how to utilize software tools to solve typical technical problems. Upon successful completion of the first course, students are able to develop and apply procedural software tools. Successful completion of the second course adds object oriented programming tools to the student's capabilities. The third course introduces students to classical data structures and advanced data types and illustrates their use in building elegant and efficient solutions for mathematical, scientific and engineering problems. The three courses contribute to an overall curriculum that places emphasis on the design of real time embedded systems and microprocessor control applications.

The three course sequence is also intended to provide introductory knowledge and preparatory experience for courses taken later in the student's program of study. These later courses include titles such as Embedded Systems Design, Digital Systems Design, Software Engineering, Computer Architecture and Networks and Distributed Systems

Content for Courses

Most programming courses contain a commonly-accepted set of topics which provides a foundation upon which to build a new course. This set of topics leads to the development of a set of activities (readings, lectures, problem solving exercises, laboratory assignments and testing) which promote student learning.

With this in mind, the foundation topics for the first course in the sequence were selected to include variables, data types, operators, structures (sequential, selection and repetitive), functions and arrays. Students are expected to learn not only the syntax and grammar of the C++ language associated with these topics, but the assignments and laboratory exercises encourage them to become familiar with when, where and how to apply these programming elements to solve technical problems.

The second course in the sequence includes topics associated with the use of the C++ language to construct object oriented code. Representative topics are data hiding, inheritance, polymorphism and code reuse. Students are expected to learn class construction, overloading methods, pointer notation along with aspects of acceptable software engineering and the relationship between software code and hardware components.

The third course covers foundation topics such as recursion, stacks, queues, trees and data lists as applied to sorting and searching problems. Project management tools and techniques are briefly described and illustrated through the use of case studies. Solutions to more complex problems via numerical, graphic and statistical methods allow the student to demonstrate an ability to work in groups to bring together all of the C++ programming and software development skills learned throughout the sequence. In this course, team projects are used as a replacement for individual laboratory assignments.

Text Selection

It is imperative that the instructor use a text which completely covers the key topics for each course. Additionally, the text must be technically accurate, readable, concise and hopefully filled with example programs and graphics (to accommodate visual learners). Finding a textbook that meets these requirements and emphasizes technical problem solving is not an easy task.

Generally texts address the use of programming and software code development to solve business problems, display and graphics problems or engineering problems. The text selected for the first two courses of the sequence is based upon the solution of scientific and engineering problems (5). In addition to the basic C++ language features for procedural and object oriented programming, the selected text provides examples and problems which represent real-world engineering challenges. Its style and presentation methods help students to understand the importance of writing code which can be easily modified and maintained over a period of many years. This is an excellent introduction to the principles of software engineering.

The text selected for the third course was found to be unsatisfactory when the course was first presented. It was a classical data structures text which had been converted from the C language to C++. One particular shortcoming was that many of the important program examples were not fully converted. Switching between C and C++ code proved to be confusing for students. Another text has been selected for use in the Spring 2002 quarter.

Common Elements for All Three Courses

Learning of basic programming techniques was a prime consideration in the development process. Lectures, programming assignments and laboratory problems were all selected to allow students to demonstrate their learning of grammar and syntax associated with C++ code generation.

Hands-on learning of programming skills is best learned through performance of laboratory assignments. In particular, the Laboratory problems (two per week) were selected so that students would be exposed to the use of programming structures, data structures and programming techniques (e.g. recursion) coordinated with the Lecture presentations of these topics. My observation is that one hour spent by a student in the laboratory, struggling to solve a problem, is a better learning experience than several hours of listening and watching during a lecture.

Testing must be designed to emphasize student learning of programming tools, skills and techniques. Early in each of the courses (first three weeks), quizzes and tests emphasize definitions and concepts. Multiple choice and true/false question formats support testing of the learning of conceptual material. Later in each course, tests which require writing of programs are used to assess the level at which students are learning how to solve problems and use the available C++ tools to develop solutions.

It is important to clearly set and enforce strict time schedules for assignments. I use a grade discount of 20% per day for late assignments. Even though this may sound severe, it assists students not to procrastinate and ultimately fall behind. It seems like I am forever reminding students that excuses for late submissions will not be tolerated very long when they enter an industrial position either as a co-op or a graduate.

Elements of the courses stress the importance of structured programming formats. The use of identification comments at the start of each program and the inclusion of user-friendly aspects in all code count for up to 20% of the grade for a program. Software engineering aspects are considered with regard to code maintenance over the entire software life cycle (cradle to grave). The cost of software development has become the largest component of many typical industry projects. Learning to abide by and strictly adhere to basic principles of software engineering, can contain and cap the total life cycle cost of code ownership.

Each laboratory or project assignment requires the submission of a professional laboratory report. A professional report is one with a neat, consistent format which

includes sections which address the five-step problem solving approach (analyze, design, code, debug and test). The report must use proper English grammar, punctuation, spelling and sentence structure. Lecture material supports the development of proper documentation and its importance to software maintenance. I urge students not to submit any written items of lesser quality than what they would give to the top executive of the firm where they work.

Each student in each of the courses is assigned to a weekly laboratory session. The labs are the most important aspect of each course; therefore, attendance is **required**. Each laboratory assignment is designed to promote learning of the C++ language features covered in that week's lectures. Assignments consist of two tasks – the pre-laboratory problem and the main problem. The solution for the pre-laboratory problem is to be turned in at the start of a laboratory session. The solution to the main task is to be started during the lab session when the faculty member is available for assistance and it is due before the start of the next lab session. Programs and laboratory reports are prepared as electronic submission.

I use the following check list for correcting and grading laboratory assignments and tests.

Internal Program Code Format and Style

- Does the program include a comment block with the author's name, program creation data, and anti-plagiarism statement?
- Are variable names descriptive and appropriate?
- Are comments used to describe each program variable?
- Is white space used to improve the readability of the program?
- Are comments used appropriately to describe the program flow?
- Are indents used properly to denote IF/FOR/WHILE/ELSE/SWITCH statements?
- Are brackets aligned properly?
- Is the user interface visually pleasing?
- Are the on-screen directions or explanations clear and concise?
- Is the user able to crash the program with unusual or unexpected inputs?

Accompanying Documentation

- Is the Laboratory Report documentation neat, professional?
- Are all the requested topics included and clearly formatted with appropriate headings?
- Is the programming log realistic and accurate?
- Does the Laboratory Report discuss the lessons learned by the student.

Mostly in the second and third courses of the sequence, laboratory assignments and lecture materials stress the importance of software configuration and it relation to efficient hardware operation. This is accomplished by constantly reminding students of the hardware activities which take place when specific C++ statements are executed. For instance, the declaration of a variable of a specific data type results in a certain number of

bytes of memory being reserved for storage of values. Memory limitations are easily demonstrated when students are learning recursive coding methods.

Student-to-Student Learning

The purpose of the laboratory assignments in these courses is for students to learn more about programming computers than can be covered in the course lectures and readings. Learning and knowledge retention occur in the doing of these projects. The labs are learning exercises, not tests of what students know. Therefore, students are encouraged to talk with other students about approaches to solving the assigned problems. However, all written work and program development is to be done individually. All computer programs and reports must be individually developed. It is stressed that copying work from other students will result in all involved students receiving a grade of 0 for the assignment and the possibility of a failing grade in the course.

Each syllabus for these courses contains the following warning.

> "Plagiarism" (from a Latin word for "kidnapper") is the presentation of someone else's ideas or words as your own. Whether deliberate or accidental, plagiarism is a serious and often punishable offense. **Deliberate plagiarism includes** copying a computer code from a source and passing it off as your own or handing in as your own work a program you have bought, had a friend write, or copied from another student or a published source.
>
> Each RIT student is expected to maintain high standards of honesty and ethical behavior. All individual assignments must be completed individually. Therefore, it is required that you add these comments at the top of each program you submit for course credit.
>
> ```
> //--------------------------------------------------------------------------------
> //      On My Honor……
> //      I have not copied program code from others or from published sources.
> //      The code for this assignment was created by me and is original.
> //--------------------------------------------------------------------------------
> ```

Grades and Evaluating Student learning

The courses were developed for delivery in a three-quarter academic year format. This allows an eleven week time period for each course. The courses were developed to include several overnight assignments, two quizzes, a midterm examination and a two hour final examination.

Quizzes normally allow 30 minutes for each student to prepare a written C++ program which provides the solution to a single problem. An alternative test format consists of multiple choice and true-false type questions which measure the degree of learning of

definition type material including grammar and syntax facts which are covered in lecture and laboratories.

The midterm and final examinations normally consist of writing code which allows students to demonstrate an ability to solve problems using C++ programming techniques. The final exam covers material for the entire course.

Assignment	Quiz One	Midterm	Quiz Two	Final Exam	Laboratories
Weight	10%	20%	10%	30%	30%

Assigning heavy weights to the laboratories, midterm and final examinations emphasizes the importance of problem solving using C++ tools. Lectures include examples where the solution is presented in a five step (analyze, design, code, debug and test) approach. All problems (lecture examples, lab assignments and examination questions) are selected to illustrate common situations encountered by scientists and engineers. Problems span a variety of ET disciplines including electrical, computer, civil and packaging.

Distance Learning Versions

After development and delivery of each course in the classroom environment, a distance learning version was prepared for delivery in an asynchronous manner over the Internet. The distance learning versions were heavily based on the philosophies and approaches used for the classroom version. Conversion to the virtual environment required that materials, lectures, assignments and testing methods be evaluated with respect to the levels at which they supported student learning. The major changes were made to encourage increased student to student interactions and remote testing.

A discussion board was made part of each distance leaning course. A percentage (~25%) of each student's final, overall grade was tied to a requirement to post significant messages related to the topics being covered during each week. Initial student reaction to this work was negative. But, within a few weeks, most students could see the value of student-to-student learning. Early in the course, I remind the students that we learn from each other.

Conclusion

This article has presented activities and experiences related to development of a three-course sequence of technical programming. The three courses contain elements inserted to specifically meet current and future needs of ET students. Besides the grammar and syntax elements of the C++ language, the courses emphasize technical problem solving, software engineering practices, project management and team aspects and good communications skills. Key elements and the overall philosophy for the course sequence are examined with an eye toward ensuring that a focus on student learning is maintained.

Bibliography

1. Committee on Information Technology Literacy, "Being Fluent with Information Technology," National Academy of Sciences, Washington, DC, Book and Web Site NSF Contract No. CDA-9616681, 1999.
2. ACM Curriculum Committee on Computer Science, "Curriculum 78: Recommendations for the Undergraduate Program in Computer Science," Communications of the ACM, Vol. 22, pages 147-166, 1979.
3. Technology Accreditation Commission for Accreditation Board for Engineering and Technology (ABET), "CRITERIA FOR ACCREDITING ENGINEERING TECHNOLOGY PROGRAMS Effective for Evaluations During the 2001-2002 Accreditation Cycle," http://www.abet.org/images/Criteria/tac_criteria_b.pdf
4. Undergraduate Bulletin 2000 – 2001, Rochester Institute of Technology, Volume 15, No. 1, page 175, July 11, 2000.
5. *C++ for Engineers and Scientists*, Gary J. Bronson, PWS Publishing Company, ISBN: 0-534-95060-4, (1998).

Author Biography

Anthony Trippe is a generalist with a BS in chemistry (1966), an MS in Mathematics and Computer Science (1972) and a Doctor of Business Administration (1982). He is an assistant professor at the Rochester Institute of Technology in the Electrical, Computer and Telecommunications Engineering Technology Department. He teaches technical programming and computer technology courses in the classroom and over the Internet for RIT. He is also an adjunct faculty member at the University of Phoenix Online Campus where he has been facilitating courses for over four years. His UOP courses include graduate and undergraduate Project Management, Operating Systems, Computer Architecture, Statistics, Strategic Planning and Computer Programming. Much of the information presented in this paper is derived from his personal experience as a teacher and facilitator in both the classroom and on the Internet.

Contact: Dr. Anthony P. Trippe
Rochester Institute of Technology
Computer Engineering Technology
78 Lomb Memorial Drive
Rochester, New York 14623

E-Mail: aptiee@cast-fc.rit.edu
Phone: (585) 475-6537
URL: www.rit.edu/~aptiee