In 1981 Rentsch wrote "Object oriented programming will be in the 1980s what structured programming was in the 1970s" [Rentsch 81]. The validity of this assertion is supported by the proliferation of books and journals devoted primarily to the subjects of object oriented software development and object oriented programming languages, particularly C++. In addition the commercial software development community has made widespread use of the object oriented paradigm. Now in the 90's these circumstances would seem to require that computing science educators find a place in the curriculum to teach the concepts of object oriented software development(OOSD). Temte has persuasively encouraged us to begin teaching the object oriented paradigm [Temte 90]. The question to be answered is, of course, "What should we teach and where should we teach it?"

Although several new texts designed for first year courses include discussions of object oriented as well as task decomposition approaches to program development, we at ECU believed that understanding object oriented programming concepts required more maturity than that possessed by our freshman students. We decided against integrating object oriented principles into our existing courses preferring to gain experience teaching OOSD before considering a reorganization of our curriculum. Thus we decided to offer a separate special topics course in Object Oriented Programming open to students who had completed our CS2 (Pascal) course. The course was first offered during the summer of 1991 and had an enrollment of 14 including four students having only the minimum prerequisite and one student who graduated at the end of that summer.

Deciding what to teach was more challenging. Some attention needed to be given to most steps of the object oriented development lifecycle. We wanted to examine the development of object oriented programs starting with analysis and concluding with implementation. Testing and maintenance were omitted due to lack of time to deal with them. The language chosen to implement the programs was C++. This choice was made in part because that language appears to be the most widely used object oriented language (particularly in industry) and because a compiler was available for our hardware and our Unix operating system.

Seven major objectives were established for the course. These objectives covered both the implementation language and the concepts used in the development of programs using object oriented methods.

To meet these objectives the students were to:

1. Learn the syntax and semantics of the C++ programming language.
2. Experience writing "traditional programs" using C++.
3. Experience writing programs in C++ using object oriented techniques.
4. Understand what is meant by object oriented analysis (OOA) and object oriented design (OOD).
5. Learn what are appropriate techniques for carrying out OOA and OOD.
6. Learn what notations are available for representing OOA and OOD.
7. Examine how C++ compares as an object oriented language with other object oriented languages such as Object Oriented Pascal and Oberon.

The first three objectives are clearly language specific. The approaches used to meet them were expected to be similar to these used in teaching the syntax and semantics of any language. Objective 2 may require additional explanation. The encapsulation features of C++ allow the generation of true abstract data types (ADT) not possible using traditional (K & R or ANSI) C. In this regard (and many others) C++ is truly a "better C".

The last four objectives were viewed as being language independent. They could exist essentially unchanged regardless of the chosen implementation language (except for the substitution of that language for C++ in objective 7). The study of object oriented analysis and design independent of a specific programming language has been recommended by Schmucker [Schmucker 86]. We considered it undesirable for this course to tie the concepts of object oriented software development to a single programming language. The principles of OOSD transcend any single language and a student should recognize by the way the principles are taught that they can be applied using C++ or Smalltalk or Object Oriented Pascal or CLOS or any other language including object based and conventional languages with appropriate modifications.

What approach should be used to present these separate sets of objectives? We decided to treat the course initially as though it was two distinct courses. The summer term course met for 60 minutes a day, five days a week for eight weeks. Three days a week (MWF) the semantics and syntax of C++, were discussed. Two days a week (TTh) the concepts of object oriented development were presented in a language independent manner. We anticipated that after about four weeks these two threads would come together and the students would be in a position to use object oriented principles to develop and implement programs in C++.

Two texts were recommended. The book by Dewhurst and Stark was adopted for the C++ portion of the course [Dewhurst 89]. Booch was used for the object oriented principles [Booch 91]. The text by Graham which was published after the textbook adoption date was actually used by the instructor as the major resource to teach C++ and was purchased by a number of the students [Graham 91]. Booch's book was placed on reserve in the library and the instructor's copy was available for loan. Several students also purchased personal copies. Five programs were assigned dealing with C++ I/O and variables, C++ control structures, classes, overloaded operators and functions, and class inheritance and polymorphism. A midterm and a final exam were given to further evaluate the student's theoretical understanding of the concepts.

Since a knowledge of C was not a course prerequisite, objective one was met by teaching C++ syntax and semantics to include the elemental control structures and operators common to C and C++ as well as those aspects of C++ not represented in C. The latter includes the stream I/O and struct modifications in particular. Students were required to use C++ constructs rather than equivalent C constructs when writing their programs and were instructed in the C++ culture in an attempt to break experienced K & R C coders from their traditional
ways of programming [Stroustrup 87]. However, C++ is a larger language than C. Edelson suggests than C++ is fifty percent more complex than C in part because it has more key words (42 to 29) and more operators (47 to 44) [Edelson 89]. Consequently more than twelve class periods were required to discuss the basic syntax and semantics of the language.

The approach used to meet objective 2 was to focus on C++’s built in support for ADT’s. The struct in C++ can include functions as well as data types and is therefore ideally suited for use in defining ADT’s. The addition of the key word private when defining the data types of the structs provides for true encapsulation. The concept of the ADT was new for those students who had not had a course in data structures. This objective primarily illustrated how C++ is a “better C” in terms of its ability to enforce concepts of programming in the large.

The third objective was met by introducing the C++ construct of classes as well as the C++ implementation of the object oriented principles of inheritance, operator and function overloading and polymorphism. At this point in the course sufficient information had been presented about the essential basis of object oriented programming in an implementation independent fashion so that the students were able to concentrate only on the C++ implementation methods for these concepts without having to also understand the principles themselves.

The fourth objective was met in an implementation independent fashion using Booch as the major resource. First the essential characteristics of object oriented problem decomposition were discussed and contrasted with the functional decomposition approach. This presentation followed Booch’s description of the object model and focused on four principles of OOSD: data abstraction, encapsulation, inheritance and polymorphism. This material was relatively difficult for the students to grasp due to its highly abstract nature.

The fifth objective was approached using the concepts of objects and classes as the primary tool in carrying out object oriented analysis and design. The concept of class inheritance was discussed extensively as was the difference between an object and a class. Also emphasized was the requirement to use these concepts while carrying out OOA to create a true object oriented problem decomposition. OOA requires that the analyst identify the objects in the problem domain and determine the relationships between these objects in order to define the classes that are significant.

The sixth objective was discussed only briefly due to time constraints. The OOA and OOD notations of Booch and of Wasserman [Wasserman 91] were presented and the students were required to use one or the other in the description of the design of their final program. No attempt was made to examine the relative merits of these design notations.

The last objective was not met. The intent was to briefly present the approach used to implement object oriented programming principles in two other procedural object oriented languages, Object Oriented Turbo Pascal and Oberon. Haiduk’s book was to be used a reference for Pascal [Haiduk 90] and several papers as references for Oberon [Pountain 91, Wirth 88]. These languages were chosen because our students would have some familiarity with them (Oberon is derived from Modula-2) and because implementations of these languages were available. Since C++, Object Oriented Turbo Pascal and Oberon are all descendants of non-object oriented languages, their different methods of implementing the object oriented paradigm were expected to provide the students with a broader perspective on object oriented programming.

What lessons can be derived from this first experience in teaching OOSD? First and foremost there was too much information to be covered in eight weeks. Even in a regular fifteen week semester it would be too much of students to expect them to master a new
language (the complexity of C++ and the different C++ culture makes it sufficiently unlike C to be considered a different language) and a new software development paradigm. A better approach would be to separate objectives one and two from the rest and teach them as a lower level course. An appropriate place to present these principles could be in a data structures or a CS2 course that emphasizes abstract data types. The remaining objectives can be presented to students in an upper level course after they have achieved the additional maturity necessary to master OOSD. Such an approach has also been suggested by Northrup [Northrup 91]. She has proposed a five part taxonomy of software design paradigms with object oriented as the last and most complex.

Thus our experience at ECU leads us to believe that a complex object oriented language such as C++ cannot be introduced to students at the same time as they are learning the principles of OOSD. Teaching the syntax and semantics of C++ in a lower level course allows students to gain knowledge of the language prior to using it for OOSD. The object oriented paradigm is too complex to be introduced in a software development course if the students are required to use OOSD in their project for this course. Consequently, OOSD would appear to be taught most appropriately in an upper level course devoted solely to its content and presenting the principles of object oriented development to students who are already familiar with the chosen implementation language.

BIBLIOGRAPHY