Course Outline for Computer Science 15
OBJECT-ORIENTED PROGRAMMING METHODS

Catalog Description:

15 – Object-Oriented Programming Methods 4 Units

Object-oriented programming methods employed to design, program, test and document intermediate level problems. Includes strings and string objects, multidimensional arrays, pointers, dynamic allocation, classes, overloaded functions, inheritance and polymorphism, introduction to linked lists. Designed to satisfy Association of Computing Machinery (ACM) guidelines for CS I as required for Computer Science and related transfer majors. Prerequisite: Computer Science 14 (completed with a grade of "C" or higher). Strongly Recommended: Mathematics 20 (completed with a grade of "C" or higher). 3 hours lecture, 3 hours laboratory.
[Typical contact hours: lecture 52.5, laboratory 52.5]

Prerequisite Skills:

Before entering the course the student should be able to:

1. follow the procedures used in the appropriate college computer laboratory to sign in and out, and to write, edit, compile, run and debug programs;
2. demonstrate steps involved in program development;
3. write simple C++ data types in programs and apply how they are represented in the machine;
4. write C++ expressions using selected operators, and apply the rules of precedence used in their evaluation;
5. apply the structured programming constructs: sequence, selection and iteration;
6. perform elementary interactive input and output operations;
7. code void and value-returning functions with value and reference parameters and use them in a program;
8. define and use the structured C++ data types: array, string, struct in applications drawn from mathematics, the sciences, and other areas;
9. use text files to record and retrieve information in elementary applications;
10. produce well-documented, user-friendly programs of short to medium length.

Expected Outcomes for Students:

Upon completion of the course the student should be able to:

1. explain the steps in the software engineering lifecycle;
2. use accepted design methods (structure charts, pseudocode, simple class diagrams) to develop and code several (six-ten) programs of intermediate difficulty and length in an object–oriented language, including programs broken into several classes.
3. adhere to style and documentation standards in writing programs;
4. use system debuggers to step into and over code, set breakpoints and watch variables;
5. identify what makes a good test data suite and use it to thoroughly test a program under development;
6. write overloaded functions and simple recursive functions, function templates (C++) or generic methods (Java);
7. define, initialize, dereference and manipulate pointers; * see note below
8. manipulate arrays using pointer notation; * see note below
9. create multi-dimension arrays dynamically using pointers (C++) or references (Java).
10. define, design and use simple classes, including at least one project that uses a class inheritance hierarchy;
11. manipulate objects of the standard class libraries such as strings, vectors and streams,
12. overload operators (C++) or override `clone()`, `equals()` and `toString()` (Java) , define and use virtual member functions to implement polymorphic behavior;
13. identify bitwise, unary and binary and conditional operators;
14. use the new operator to implement a singly linked list.

* If the course is taught in Java, C++ -style pointers should be covered in the first few weeks, as part of a transition from C++ to Java and as a segue to Java-style references.

Course Content (Lecture):

1. Program Design and Development
   a. The software engineering life cycle: Needs analysis, needs specification, algorithm design, software implementation, testing and integration;
   b. Algorithm design: modularization and use of structure charts, pseudocode, simple class diagrams, top-down vs. bottom-up methods;
   c. Use (and misuse) of global and static variables;
   d. Use of drivers and stubs in program development;
   e. How to design a test suite;
   f. On-line debuggers: stepping into and over code, setting breakpoints, watching variables;
   g. Documentation and style standards, including naming conventions for constants, variables, and classes, commenting functions, indentation.
   h. Multi-file programs
      1) Scope and lifetime of variables: local vs. global, side effects, automatic vs. static
      2) Advantages of separate compilation
      3) How to create a multi-file project
   i. Preprocessing and compilation
      1) Preprocessor directives #if, #ifdef, #ifndef, #else, #elif, #endif (C++)
      2) Conditional compilation and use in debugging
2. Simple and Structured Data Types, Operators, Control, Functions
   a. Review of simple data types: int, unsigned, long, char, float, double, bool, enumerated types
   b. C++ pointers
      1) Concept of pointer
      2) Definition and initialization
      3) Dereferencing and pointer arithmetic
      4) Array name as a constant pointer
      5) Memory management issues
   c. Expressions and operators
      1) Dereferencing and address-of operators
      2) Bitwise operators: and, or, complement, left and right shift (optional)
      3) Binary and unary scope-resolution operators
      4) Conditional operator
   d. Quick Review of Selection and Iteration control structures:
      1) For, while, do-while
      2) If, if–else, switch
      3) Break, continue, exit
   e. Use of functions in modular programming
      1) Function declarations, prototypes and calls
      2) Parameter passing mechanisms: value, reference, pointer
      3) Function templates (C++) or generic methods (Java) and overloaded functions
      4) Default arguments
      5) Inline functions
      6) Recursive functions
      7) Passing functions as parameters (optional)
   f. Arrays and Vectors
      1) Accessing array or vector elements by index, pointer or iterator.
      2) Arrays/vectors of objects
      3) Passing arrays to functions
      4) Multidimensional arrays
      5) null terminated character arrays
   g. Objects and use of pointers or references to Objects.
      1) Access elements of an object using the member selector operator.
      2) Passing and returning objects to and from functions
      3) Use of the assignment operator between objects.

        4) Nested structures
        5) One-dimensional Arrays of objects
    h. Files and interactive I/O (Review)
        1) Concept of input/output streams
        2) Sequential text files
        3) Stream manipulators
        4) How to use istream and ostream classes to change I/O direction at runtime (optional)
3. Object Oriented Programming Concepts
    a. Classes: Vocabulary and Syntax
        1) Vocabulary of objects: object, method or member function, attribute or data member, information hiding or encapsulation, object instantiation, client
        2) Constructors: Definition, role of, overloaded, default, copy
        3) Destructors (C++) or garbage collection and `finalize()` (Java)
        4) Member functions: design, syntax, use, when to use const
        5) Operator overloading (C++) or overriding `Object` methods `equals()`, `clone()` and `toString()` (Java)
        6) Friend functions: Definition, when to use.
        7) *This* pointer and its relationship to the current object
        8) Private, protected and public members of a class, static class members
        9) Inheritance: Base and derived classes
        10) Composition vs. Inheritance
        11) Virtual and pure virtual (abstract) functions, abstract classes and interfaces
        12) Polymorphism: Implementation via virtual functions
    b. Introduction to Unidirectional Linked Lists
        1) 1-D Array as Abstract Data Type list
        2) Dynamic array (implemented with new)
        3) Singly linked list Abstract Data Type
        4) Defining a linked list class
        5) Advantages/disadvantages of using a linked list instead of list as array
        6) Recursive traversal and other simple list operations

Course Content (Laboratory):

1. Designing and implementing moderate-length programs
2. implement various string library functions e.g. for tokenizing a string or locating a substring
3. Two-dimensional array problems
4. Simple classes, inheritance and polymorphism
5. Operator methods and friend functions
6. Simple recursive problems, e.g., maze traversal, solve a Sudoku grid
7. Testing issues and techniques

Methods of Presentation:

1. Lecture
2. Discussion
3. classroom demonstrations
4. Hands-on exercises in the laboratory

Assignments and Methods of Evaluating Student Progress:

1. Typical Assignments
    a. Create a project that includes multiple classes implemented in separate files along with a driver program.
    b. Create an inheritance hierarchy for the classes Quadrilateral, Trapezoid, Parallelogram, Rhombus, Rectangle, and Square. Make Quadrilateral the base class.

2. Methods of Evaluating Student Progress
    a. Midterms
    b. Final examination

c. Assigned programs
d. Homework assignments

<u>Textbook(s) (Typical)</u>:

Starting Out with Java™: From Control Structures though Data Structures, first edition, Tony Gaddis and Godfrey Muganda, Pearson Prentics Hall 2007.

Starting Out With C++ From Control Structures through Objects, sixth edition, Tony Gaddis, Pearson Prentice Hall, 2009

<u>Special Student Materials</u>:

Portable storage device such as a USB flash drive

Carol Conway and Maurice Ngo November 8, 1995

Revised October 7, 1997 Carol Conway
Revised October 7, 1998 Carol Conway
Revised August 22, 2001 Carol Conway
Revised October 2003, Wanda Wong
Revised October 5, 2009 Keith Mehl
Revised November 6, 2010 Jonathan Traugott and Wanda Wong

Effective F2011