CS 1120 – COMPUTER SCIENCE II
SummerII Term – 2005

Instructor: Dr. Ajay Gupta
Office Hours: See the CS 1120 web page
http://www.cs.wmich.edu/~gupta/teaching/cs1120/sumII05/

Catalog Description of CS 1120
This is the standard Computer Science II course using the C++ computer language. The emphasis is on designing and programming object-oriented computer solutions to problems, as well as on the data structures used for this purpose. An introduction to the analysis of algorithms is made. Students must register for both a lecture section and a laboratory section.

Prerequisites
By Courses: CS1110 – Computer Science I or equivalent with a grade of C or better (prerequisite); Math1220 (old Math122) or Math 2000 (old Math 200 - corequisite)
By Topic: Basic concepts of high-level language programming – conditional structures; looping structures; arrays; program logic – to solve problems; Basics of object oriented programming - be able to create and use elementary objects; C++ language for both procedural and introductory object oriented programming; Basics of the software life cycle; Validating quality of software produced; Introductory sorting and searching algorithms; Algorithms for elementary problem solutions; Documenting programs effectively and efficiently.

Objectives
• Lean about each phase in software life cycle.
• Understand the concepts of abstract data types.
• Understand the concepts of recursion, inheritance, and polymorphism.
• Learn basic mathematical techniques for analyzing algorithm complexity.
• Learn common data structures

Performance Objectives
• Student will be able to write well-designed and well-documented programs.
• Student will be able to use recursive solutions to solve complex applications.
• Student will be able to use abstract data types and hide implementation from users.
• Student will know the data structures such as linked list, stack, queue, tree, table, graph and the operations performed on these data structures.
• Student will know how to derive new classes by inheritance and polymorphism.
• Student will know different sorting algorithms and be able to analyze their efficiencies.

Text

Required


Recommended


One of these:


Important Dates

June 29, 2005: Add/drops without any penalty.
July 21, 2005: Last day to withdraw.

Grading

During the term there will be two in-class exams – one around midterm week and one around finals week. Laboratory assignments will be given in the regularly scheduled laboratory. Pop-quizzes may be given at anytime in lab or lecture without prior notification. Your grade will be computed from your performance on these components using the following weights:

Exam I (Tentative 7/18/05) 25%
Exam II (Tentative 8/10/05) 25%
Labs 40%
Pop-quizzes 10%
PMT (8/9/05) 0%

Following grading scale will be used.
A 90; BA 85; B 80; CB 75; C 70; DC 65; D 60;

Missed Exams

If you miss an exam (Exam I or II), the decision as to whether or not it is made up and how it is made up will be made on an individual basis. To be excused there must be significant circumstances beyond the student’s control. Generally this will require documentation, such as a doctor’s note in the case of illness. Normally, if the absence
from an exam is excused, the average of your other exams will be used to replace it. If it is unexcused, 70% of the average of other exams will be used. At most one missed exam, excused or unexcused, will be made up in this way. You must inform the instructor before the exam if there are circumstances beyond your control that will cause missing an exam.

**Laboratory**

Students taking this course are required to register for a lab (recitation) section. Lab grades are based on student performance on programming assignments and quizzes. Additionally, the Programming Skills Mastery Test (PMT) is given in the lab.

**Programming Assignments**

Lab assignments will be given on a regularly scheduled basis. Many of these assignments will need to be worked on outside of the regular scheduled labs. Each assignment will have a due date/time. For each day an assignment is late, 10% of the total possible points for the assignment will be deducted. If an assignment is more than 10 days late, it is no longer worth any points. Weekends and holidays are all counted when calculating lateness. No assignments may be submitted after 11:59 PM on Wednesday, August 10, 2005. By this time all work should be complete and submitted.

**Quizzes**

There will be regular quizzes given in the lab. Additionally, pop-quizzes may be given at anytime in the labs or lectures without prior notification. If you miss a quiz for any reason, you will receive a 0 on it.

**PMT**

During the August 9, 2005 lab session, students will be given the Programming Skills Mastery Test (PMT). The test will consist of a short programming problem. Students must program the solution in an essentially complete and correct form in the allotted time. This problem must be solved within the allotted time to earn a passing grade in the course. Students that fail the PMT on their first attempt will be allowed to take it one additional time with a different problem during the lab on August 11, 2005.

**Special Policies**

Producing competent programmers is a primary goal of this course, and therefore a minimum performance in lab is required for students to pass the course.

- You must pass the lab with at least 60% of the total possible lab points in order to pass the course regardless of exam scores.
- You should strive to complete all assignments. In order to pass the laboratory, you may have at most two assignments incomplete. Even if an assignment is so late that the credit would be 0, it can still satisfy the completeness policy if it is completed and submitted.
- You must pass the PMT to pass the course.
Incomplete Grades

Please note that the incomplete grade - I - is intended for the student who has missed a relatively small portion of work due to circumstances beyond his/her control. In general, performance on work done must be at a level of C or better in order to qualify for an incomplete. An I grade will not be given to replace an otherwise low or failing grade in the class.

Academic Honesty

The following statement has been approved and distributed by the Western Michigan University Faculty Senate:

You are responsible for making yourself aware of and understanding the policies and procedures in the Undergraduate or Graduate Catalog that pertain to Academic Integrity. These policies include cheating, fabrication, falsification and forgery, multiple submission, plagiarism, complicity and computer misuse. If there is reason to believe you have been involved in academic dishonesty, you will be referred to the Office of Student Judicial Affairs. You will be given the opportunity to review the charge(s). If you believe you are not responsible, you will have the opportunity for a hearing. You should consult with me if you are uncertain about an issue of academic honesty prior to the submission of an assignment or test.

Unless otherwise told, you may not bring aids to exams. Submission of another person’s work in part or whole is not permitted. Learning can certainly occur with discussion of class material and assignments with other students, and we will be doing considerable collaborative activity, but at all times take care that you don’t represent the work of another as your own.

- If you are copying another’s work in part or whole, either by hand or electronically, you are going too far.

- If two or more people are working so closely together that the outcomes, particularly on significant portions of computer programs, are essentially line-by-line the same in logical structure, they are going too far.

- You should not give your completed work to someone else or accept another’s completed work to “review or look at” in either hardcopy or electronic form. This too easily facilitates copying.

If you are found responsible for violation of academic honesty in the course, you will receive a penalty up to and including an E grade in the class.

Course Topics

Programming Topics

- Coding Standards
  - Declarations vs. definitions; Headers and code; Indentation;
  - Commenting/documentation
• **Compilation Process**
  Preprocessor; Compilation; Linking

• **Classes**
  What is a **class**? Terminology: objects, classes, instances; Examples of classes from modeling; Member functions/methods; Member variables; Access functions (& pass by reference); Constructors and destructors; Scope; Constant objects, functions, arguments; Static members; Friend functions and classes

• **Stream I/O**
  Stream classes and objects; Basic operations on streams; getline(), get(), etc.; Detecting the end of input; Manipulating streams (iomanip); Using text file streams

• **Operator Overloading**
  Why overload? How overloading works; Operators and friends; Unary and binary operators; Comparison operators; Other operators: cast and index

• **Pointers and Dynamic Memory**
  Pointer concepts; References to this; Allocating dynamic memory; Multiple references to a single item; Common errors and debugging

• **Inheritance and Polymorphism**
  Has-a vs. is-a designs; Subclassing; Access: **public**, **private**, and **protected**; Abstract base classes; Virtual functions; Late binding

• **Templates**
  Function templates; Class templates; Standard Template Library (STL)

• **Tools**
  **Make** and Makefiles; **XEmacs** - a programming-oriented editor; **DDD** - a GUI debugger; Command-line argument handling and **getopt()**; **rand()** and **srand()**; **time()** and **gettimeofday()**

**Concepts**

**Program Design Using ADTs**
Cohesion and coupling; Abstraction - functional and data; Problem statements and the noun/verb division; Producing a basic design document; Moving from design to code

**Recursion**
Recursive functions: fact and fib; Recursive algorithms: binary search; Using the system stack: reversing a LL or char* string; Problem-solving and traversing search spaces

**Array Data Structures**
Arrays of objects; Arrays of pointers to objects; Dynamic arrays

**Linked Lists**
Insertion: beginning, middle, end; Deletion: beginning, middle, end; LL as a modeling tool (trains, etc.)

**Stacks**
Uses and push/pop; Implementation as a LL; Sample Algorithms: postfix, delimiter match
**Queues**
Uses and enqueue/dequeue; Sample algorithms: palindromes, string to int conversion, simulation

**Program Analysis**
Analytic vs. experimental methods; Big-O concepts; Basic code analysis: loop structures; Analysis of bubble, insertion, and selection sorts; Analysis of binary search, towers of hanoi, quicksort, and mergesort

**Algorithms**
Linear and binary search; Bubble, insertion, selection, quick, and merge sorts; Postfix evaluation; Delimiter matching; Reversing linked lists; Palindrome recognition