CSC 222: Computer Programming II

Spring 2004

See online syllabus at: www.creighton.edu/~davereed/csc222

Course goals:

- to know and use basic programming tools for object-oriented problem solving (e.g., classes, encapsulation, data hiding, templates)
- to appreciate the role of algorithms and data structures in problem solving and software design (e.g., OOD, searching and sorting, recursion, stacks, queues, linked lists)
- to be able to design and implement a program to model a real-world system, and subsequently analyze its behavior.
- to develop programming skills that can serve as a foundation for further study in computer science

221 vs. 222

221: programming in the small

- focused on the design and analysis of small programs
- introduced fundamental programming concepts
  - ✓ variables, assignments & expressions
  - ✓ input and output
  - ✓ functions & parameter passing (by-value & by-reference)
  - ✓ control structures: if, if-else, while, for
  - ✓ arrays
  - ✓ using existing classes: string, ifstream

you should be familiar with these concepts (we will do some review next week, but you should review your own notes & text)

222: programming in the medium

- focuses on the design and analysis of more complex programs programs
- will introduce more advanced programming concepts & techniques
- greater emphasis on problem decomposition, code reuse & modifiability
  - ✓ classes, composition, data structures
  - ✓ searching & sorting, recursion, algorithm efficiency
  - ✓ vectors, stacks, queues
  - ✓ pointers, dynamic memory, linked lists

When problems start to get complex…

…choosing the right algorithm and data structures are important

- e.g., phone book lookup, checkerboard puzzle

- must develop problem-solving approaches (e.g., iteration, recursion)
- be able to identify appropriate data structures (e.g., vector, linked list, stack, queue)

…code reuse is important

- designing, implementing, and testing large software projects is HARD
  whenever possible, want to utilize existing, debugged code

- reusable code is:
  clear and readable (well documented, uses meaningful names, no tricks)
  modular (general, independent routines – test & debug once, then reuse)

Object-oriented programming

OOP is the standard approach to software engineering

philosophy: modularity and reuse apply to data as well as functions

- when solving a problem, must identify the objects involved
  e.g., banking system: customer, checking account, savings account, …

- develop a software model of the objects in the form of abstract data types (ADTs)
  an ADT is a collection of data items and the associated operations on that data
  in C++, ADTs are known as *classes*

OOP stressed ADTs in order to

- hide unnecessary details (programmer doesn't have to know the details of the class in order to use it)
- ensure the integrity of data (programmer can only access public operations)
- allow for reuse and easy modification (can plug classes into different applications)

In-class exercise

PAPER FOLDING PUZZLE: if you started with a regular sheet of paper and repeatedly fold it in half, how many folds would it take for the thickness of the paper to reach the sun?

what information do you need (e.g., distance of sun)?

what data values do you need to store and update?

what is the basic algorithm?

pseudo-code:

```
INITIALIZE THE PAPER THICKNESS;
INITIALIZE THE FOLD COUNT TO 0;

while (THICKNESS < SUN DISTANCE) {
  DOUBLE THE THICKNESS;
  INCREMENT THE FOLD COUNT;
}

DISPLAY THE FOLD COUNT;
```