CSCI 200 Exam 2 Learning Outcomes

Course Learning Outcomes

1. Design an algorithm to solve a problem by breaking the overarching problem into smaller modular components using abstraction and object-oriented design with inheritance.
2. Translate the algorithm into a program using proper C++ syntax and fundamental programming constructs (e.g. control structures, I/O, classes).
3. Recite & apply frequently used Linux command line commands and compile a program using a command line build system.
4. Diagram memory usage, dynamically allocate & deallocate objects at run-time using "The Big Three," and trace the call stack of a program's run-time.
5. Define "Big-O" notation, list complexities in increasing order, and analyze an algorithm to compute its run-time performance & memory complexities

Module Learning Outcomes

Algorithms and Data Structures (ADS)

- Define Big O Notation and recite the dominance relations.

C++ Programming (C++)

- Discuss the differences between programming languages, specifically Python & C++
- Create a Hello World program, construct a simple interactive application, and build the program via the terminal.
- List C++ primitive data types and explain the appropriate use of each data type.
- List & identify C++ arithmetic operators, translate math equations to C++, and solve arithmetic expressions.
- Discuss the effects of a statically typed language.
- Convert one data type to another.
- Recite the order of operations and evaluate an expression.
- Explain how C++ generates random numbers and write a program that generates random numbers.
- Identify C++ control structures and conclude which branch a sample program will execute.
- List C++ logic operators and evaluate Boolean expressions consisting of multiple logic operators.
- Evaluate the resultant output of a code block containing a control structure.
- Convert a program written with a for loop to a program using a while loop and vice versa.
- Identify and correct errors in program structure and logic.
- Write a program that implements the corresponding pseudocode using file streams.
- Create a program with formatted output.
- Implement & manipulate pointers to reference memory on the stack or the free store.
- Explain the following terms and how they are used (1) dot operator / member access operator (2) data member (3) scope resolution operator.
- Explain the four uses of const in our programs and classes.

Command Line Interface (CLI)

- Create a Hello World program, construct a simple interactive application, and build the program via the terminal.
- List common Linux terminal commands and choose the correct commands to work with a file system via the command line.
- Describe how a computer generates a program from code.
- Write and use a Makefile.
- Discuss the advantages of using Makefiles.

Design Elements (DE)

- Explain how C++ generates random numbers and write a program that generates random numbers.
- Identify C++ control structures and conclude which branch a sample program will execute.
- Identify C++ repetition structures and explain the following terms: looping parameter, stopping condition, and looping parameter modification.
- Explain the appropriate use and differences between a while loop, for loop, and a do-while loop.
- Discuss the design process and strategies for developing good code.
- Identify and correct errors in program structure and logic.
- Implement various techniques to trace & debug a program.
- Discuss the pros/cons of the various debugging techniques.
- Recite the six steps to properly use a file stream for reading or writing.
- Explain the two ways to open a file for writing.
- Identify the parts of a function.
- Explain the difference between a parameter and an argument for a function. Discuss what can be returned from a function and what a void function is.
- Explain the meaning of the DRY principle and appropriate uses for functions.
- **S**OLID: Discuss how functions contribute towards the Principle of Single Responsibility.
- Define and implement a procedural programming style.
- Explain the difference between pass-by-value and pass-by-reference. Draw a diagram of how each stores its parameters in memory.
- Create a program that validates user input and removes the need for a cooperative smart user.
- Define REPL and perform read operations conforming to common input paradigms.
- Explain the difference between a function prototype and a function implementation. Discuss the pros/cons of separate implementations.
- Implement various techniques to trace & debug a program.
- Define an overloaded function and recite common usages for overloaded functions.
- Construct a program that accesses an element in a vector, returns the length of a vector, changes the length of the vector, and other vector operations.
- Construct a program that accesses an element in a string, returns the length of a string, changes the length of the string, and other string operations.
- Compare and contrast Procedural Programming with Object-Oriented Programming
- Discuss the advantages & disadvantages of POP & OOP

Memory Management (MM)

- List C++ primitive data types and explain the appropriate use of each data type.
- Explain how values are stored in memory, how the values are interpreted differently based on data type, and list common errors that can occur with data types.
- Diagram how integer and decimal values are represented in binary.
- Convert between binary and decimal formats.
- Explain the difference between pass-by-value and pass-by-reference. Draw a diagram of how each stores its parameters in memory.
- Explain the concept of local & global scope when functions are used within a program.
- Explain the difference between the stack & the free store and the contents of each.
- Implement & manipulate pointers to reference memory on the stack or the free store.
- Diagram the memory associated with pointers and where the values lie (either in the stack or the free store).
- Diagram how pass-by-pointer works with pass-by-value and pass-by-reference in functions.
- Discuss causes of & solutions to memory leaks, segmentation faults, dangling pointers, null pointer exceptions, and other pointer related errors.
- Discuss the concept of scope within and outside a class & struct
- Explain the difference between a shallow copy and a deep copy. Implement both.

Object-Oriented Programming (OOP)

- Discuss the concept of encapsulation
- Draw a class diagram using UML to describe the structure of a class and its members
- Discuss the difference between a class and an object
- Create a class containing data members and member functions
- Compare and contrast Procedural Programming with Object-Oriented Programming
- Explain and use the following terms (1) constructors & destructors (2) accessor modifiers (3) accessor & mutator functions
- Discuss the concept of scope within and outside a class & struct
- Define, list, and implement the Big 3.
- Overload common operators and discuss reasons why operator overloading is useful.
- Explain the four uses of const in our programs and classes.
- Discuss the concept of encapsulation