COMP 2404 Midterm Review | Fall 2018

Section 1 | Basics of C++ Development

Linux Platform

types of shells

- a shell is program that allows direct access to OS
- allows users to run programs
- command line interpreter
- main shells
    - sh (bourn shell)
    - bash (bourne-again shell)
    - csh (c-shell)
- differences include
    - commandline shortcuts
    - environment variables

basic unix commands / io redirection

- pipelining is used for io redirection
    - < redirects from
    - > redirects to

types of unix platforms

- lunix
- solaris
- BSD
- HP-UX
- macOS

program building

- **compiling / linking**
    - what are they?
        - **compiling:** generate object code from source files
        - **linking:** generate executable from object code
    - what commands are used?
        - gcc -c to compile
        - gcc -o to generate executable
- **source files / object files / executables**
- **makefiles**
    - what are they?
        - text file

        - organizes compiling and linking commands
        - defines source file dependencies
    - what are they used for?
        - easy way to compile and link multiple source files

Basic Language Features

terminology

- expressions
    - sequence of operation that evaluates to a single variable
- statements
    - expression terminated by a semicolon that resolves to a single variable
- blocks
    - sequence of statements within pair of braces
- scope
    - part of program where a variable can be used
- operators
    - operands
        - variables on which operator acts on
    - **arity**
        - number of operands
    - **precedence**
        - order in which operators execute
    - **associativity**
    - order in which operators of same precedence execute
        - right-to-left / left-to-right

variables

- contains: name / value / data-type / memory address
- can be primative / user-defined / pointer

functions

- global / member functions
    - static member functions are 'global' to class
- declaration / implementation
    - declaration in header
    - implementation in source
- function design
    - reusable
    - single purpose
    - encapsulation (hidden functionality)

parameter types

- input / output / inout (input-output)

parameter passing

- by-value / by-reference / by-reference-by-pointer

references

- what are they?
    - an unbreakable binding / alias for an existing variable
- what are they not?
    - they are not a variable
    - they do not occupy memory
- how are they used?
    - in order to pass parameters by reference

Programming Conventions

naming / indentation / comments

- promotes readability
- comments block in main
    - specifying purpose of program
    - usage (command line arguments)
    - author
    - revisions
- comments before class
    - specifying purpose of class
    - descriptions of complex / critical members

Section 2 | Basics of C++ Classes

Class Definitions

binary scope resolution operator ::

1 - To access a global variable when there is a local variable with same name. 2 - To define a function in source file. 3 - To access a class's static variables. 4 - Multiple inheritance.

access specifiers

- public
    - visable to all
- private
    - visible only to same class
- protected
    - only visable to sub-classes

code organization

- header

    - class defintion
    - data members
    - member function prototypes
- source
    - member function implementations
    - static data member intialization

class interface

- how you interact with a class
- shows the set of public members
    - what users need to know
        - class name
        - public members

Constructors / Destructors

default arguements

- default parameter value
- specified in function prototype
- must be right-most in parameter list

contructors

- explicitly called
- **copy**
    - takes a reference of the same class
    - called when you initialize a variable to another object
        - but not already intialized variable
    - implicitly called when you pass by reference
- **conversion**
    - takes any reference of other than same class
- **default**
    - called when memory is allocated statically / dynamically
    - initializes data members

destructors

- implicitly called
- order of execution
    - child destructors are executed before parent

copy contructors

- what are they?
- what do they do?
- when are they called?

Memory Management

stack / heap

pointers

- what are they?
- why are they used?
    - small and fixed variable size
    - allows changes to memory outside current scope
    - avoids copying data
    - the only way of using dynamically allocated memory
- how are they used?
- operators:
    - arrow (->) / dereferencing (*) / address-of (&)
    - dot operator (.) accesses object members
    - arrow operator (->) dereferences object and then accesses object member
- differences references / pointers
- parameter passing with pointers
- static / dynamic memory allocation
    - dynamic: new / delete
- memory leaks
- 4 types of arrays
    - **statically** allocated array of **object pointers**
    - **statically** allocated array of **objects**
    - **dynamically** allocated array of **object pointers**
    - **dynamically** allocated array of **objects**

Section 3 | Basics of Object-Oriented Design

Software Eng. Overview

software development life cycle

- requirements analysis
- design
- implementation
- testing

Information Hiding

single responsibility

- objects should have only one purpose

data abstraction

- seperating class defintion from implementation

    - seperate the **what** from the **how**

encapsulation

- group data that belongs together
    - protect data from bad code
- reuse code when possible
- give data memebers only private or protected access

principle of least privilege

- protect your data
- access to runtime objects should limited

Object Design Categories

types of objects

- what are they?
- what do they do?
- what are they responsible for?
- why use them?
    - **control**
        - control program flow
        - manages how classes interact
    - **view (boundary)**
        - communication /interaction with user
    - **entity**
        - persistent information
        - information that survives termination
    - **collection**
        - *collection:* data structure to store multiple data objects of same type
        - collection class stores multiple collection objects

Documenting Design

uml class diagrams

- no collection classes
    - implied with multiplicity
- no getters / setters / ctor / dtor
- **classes**
    - attributes / operations
- **associations**
    - composition / inheritance
- **composition**
    - directionality / multiplicity

Section 4 | Essential Object-Oriented Techniques

Encapsulation

composition

- member initializer syntax
- order of ctor / dtor

constants

- objects / data members / member functions
    - object / data member: constant keyword before data type
    - member function : constant keyword after protoype
- constant functions
    - cannot modify data members
    - the only type of functions allowed on a constant object
- constant data members must be initialized before constructor body
- must use member initializer syntax

friendship

- grants access to all private / protected members
    - cannot be taken
    - does not apply to derived classes of friend class
- friend function can access all members of a class

static class members

- exists even without class instances
- global to class
- only one instance allows
- intialized in constructor
- static member functions can only access static data members

linked lists

- insertion / deletion / clean-up
- singly / doubly / tail / no-tail

Inheritance

base / derived

- all base class members are inherited
    - but access modifiers change
    - private become invisible
    - protected / public remain the same

base class initializer syntax

- call base constructor as with class initializer syntax

order of execution

- ctor / dtor
- ctor: parent to child
- dtor: child to parent