Intermediate/Advanced Computer Programming
Final Exam Review

Unix

Basic OS design

C++/OOP
Basic commands
  _ ls
  _ mkdir
  _ cat
  _ Cd

Traversing Filesystem
  _ Relative/Absolute Paths
  _ .
  _ ..
  _ /

Redirection
  _ Control standard input/output to/from a program
  _ >  <  |
Redirecting/Piping

Text terminal

Keyboard

Display

#0 stdin

#1 stdout

#2 stderr

Program
Unix

- Single UNIX Specification defined by the Open Group
- Standard definition for the Unix system API
  - Specifies requirements for a UNIX system
- Not code
  - Allows different implementations
- Contains POSIX specifications
- Defines system interfaces, headers, commands, and utilities
- [Link](http://pubs.opengroup.org/onlinepubs/009695399/)
Operating System (OS)

User Space
Applications, libraries

Kernel
Process, Device, Memory management

Hardware
CPU, Memory(RAM), Devices
C++/OOP

Building executables
  – Compiling and linking

Control structures
  – Loops
  – Conditionals

Const

Passing arguments to functions

Classes and Objects
Building an Executable

Source Code

Preprocessor

Preprocessed Source Code $\rightarrow$ g++ -E

Preprocessed Source Code

Compiler

Assembly Source Code $\rightarrow$ g++ -S

Assembly Source Code

Assembler

Object Code $\rightarrow$ g++ -c

Object Code

Linker

Executable $\rightarrow$ g++

Executable

Library
Building an Executable

Source Code

Compiler

Object Code

Linker

Executable

Library

g++ -c

g++
Errors

Compile Time
Link Time
Run Time
Control Structures

Loops
- for, while, do-while
- break
- Continue

Conditionals
- if
- else if
- else
Passing Arguments

Pass by value
Pass by reference
Pass by pointer
const

Informs compiler that a value/object should not be changed

Must initialize variable when the variable is defined

const int x = 5;
//error: uninitialized const 'x'
const int x;
Pointers

Pointer → Data
void add(int a, int b, int *result)
{
    *result = a + b
}
Pointers

Pointer & (address of)

(dereference) *

Pointer

Data

& (address of)

Data
Pointers

```cpp
int x = 5;
int *x_ptr = &x;

*x = 7;

cout << x << endl;
```
int x = 5;

[const] int * [const] x_ptr = &x;
References

```c
void add(int a, int b, int &result)
{
    result = a + b
}
```
OOP

Encapsulation
Inheritance
Polymorphism
OOP

Classes

- “blueprint” for creating an object
- source code describes the class

Object

- Instance of a class
- Have common properties, attributes, operations, behavior with other objects of the same class
Objects

Accelerate
Brake
Change gear

Gear
Fuel flow rate
Speed
Constructors

Place to *initialize* member data and to allocate other resources needed (e.g. memory, files, sockets)

When code in constructor body is called, data members are already allocated

Compiler will generate a default constructor if any constructor is not defined
Constructor

Before calling constructor

Object

Data members:

\[
\begin{align*}
x &= 9827432 \\
y &= 28374243
\end{align*}
\]
Constructor

After calling constructor

Object

data members:

x = 0
y = 0
Types of Constructors

Default
Conversion
Others
Initialization List

Initialize objects/data as they are allocated

Foo:Foo() : Initialization List {}

eengine:engine() : cylinder(8) {}
Function Overloading

Using the same name for two or more functions

- Different parameter types
- Different number of parameters
- Cannot differ only in the return type
- Cannot differ only if one parameter is passed by-value and one parameter is passed by-reference

double sqrt(double x);
double sqrt(double x, int y);
float sqrt(float x);
long double sqrt(long double x);
BigInt a = "55";
BigInt b = "7";

BigInt c;

c = a + b;
Operator Overloading

Convert operator call into a function call and overload the function call

<BigInt> + <BigInt>
Operator Overloading

Convert operator call into a function call and overload the function call

\(<\text{BigInt}> + <\text{BigInt}>\)

BigInt operator+(BigInt, BigInt);

- or -

BigInt BigInt::operator+(BigInt);
Comparison Operators

Used to compare two objects

```cpp
bool operator==(const X& lhs, const X& rhs)
bool operator!=(const X& lhs, const X& rhs)
bool operator<( const X& lhs, const X& rhs)
bool operator<=(const X& lhs, const X& rhs)
bool operator>( const X& lhs, const X& rhs)
bool operator>=(const X& lhs, const X& rhs)
```
Composition

Classes allow the programmer to define user-defined data types

- User-defined data types allow the programmer to better describe real-world data

Just as objects can contain native data types, objects can also contain other objects
Composition

Clock object

Time object

- int hour;
- int minute;
- int second;

Time class
- IncrSeconds(...);
- getTime(...);
- GetTime(...);

Clock class
- DisplayTime24(...);
- DisplayTime12(...);
Inheritance

Allows one to **reuse** and extend existing classes

**Base** class
- existing class

**Derived** class
- new class that uses the base class's code

Allows different abstractions
- Is-a-kind-of
- has-a
Inheritance

class B : public A
{
    ...
};
Protection Levels

**Public** interface
- For unrelated clients

**Protected** interface
- For clients that inherit from the class

**Private** interface
- Generally for members/friends of the class
Inheritance

Clock

Alarm Clock
Virtual Functions

Clock

display()

AlarmClock

display()
Virtual Functions

Clock c;
AlarmClock ac;

Clock *myClocks[2];
myClocks[0] = &c;
myClocks[1] = &ac;

myClocks[0].display();
myClocks[1].display();
Order of Calling Constructors

Composition
Inheritance
Initialization lists
Calling Base Class Functions

Function()
Base::Function()
Enumeration/Counting

Counting example:
1+2+3+...+n

List all permutations

List number of substrings