CISC 3110 Final Review

Rough breakdown of topics:

20% Base Linux usage (basic comamnds, emacs/vi usage)
20% g++, make (Unit 1)
30% Pointers and dynamic allocation (Unit 2)
30% Classes, Objects, and Object-Oriented Programming (Unit 2)

Study advice:

- Do exercises in the book (or from other readings).
- Review "Application Activities" questions, below (and from the midterm review
- Review your solutions to the programming projects.

Note: you should expect most of the exam questions to be very familiar. I will use a lot of the questions from the RATs and the application activities, though I may make some small changes or ask them in a different way. You should expect the midterm, like the application activities, to be a mix of different kinds of questions.

1. Which of the following "understand" C++?

a) preprocessor
b) compiler
c) linker
d) both a and b
e) both b and c

Which is the **best** sequence of commands to use to compile and run a program from the source files `source1.cc` and `source2.cc`? Both files also `#include` the header file `header.h`.

a)

```
$ g++ -Wall source1.cc
$ g++ -Wall source2.cc
$ ./a.out
```

b)

```
$ g++ -Wall -c source1.cc
$ g++ -Wall -c source2.cc
$ ./a.out
```

c)

```
$ g++ -Wall source1.cc source2.cc -o runme
$ ./runme
```

d)

```
$ g++ -Wall -c source1.cc
$ g++ -Wall -c source2.cc
$ g++ source1.o source2.o -o runme
$ ./runme
```

e)

```
$ g++ -Wall -c source1.cc header.h
$ g++ -Wall -c source2.cc header.h
$ g++ source1.o source2.o -o runme
$ ./runme
```

Assuming `helloworld.cc` is a valid C++ program (no syntax errors, etc), which sequence of commands will NOT produce a valid executable?

a)

```
$ g++ -E helloworld.cc > helloworld.ii
$ g++ -Wall -S helloworld.ii
$ g++ helloworld.s -c
$ g++ helloworld.o
```

b)

```
$ g++ -Wall -S helloworld.cc
$ g++ helloworld.s
```

c)

```
$ gcc helloworld.cc
```

d)

```
$ g++ helloworld.cc -c
$ g++ helloworld.o
```

e)

Neither a nor c will produce a valid executable.

Consider the 4 snippets of text (**1**, **2**, **3**, and **4**). These are output fragments from various phases of the compilation of `helloworld.cc`.

**1**

```
main:
.LFB971:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $16, %esp
```

**2**

```
C9><C3>Hello,
world.^@^@^@^@s^@^@^@^@GCC: (Ubuntu
4.8.4-2ubuntu1~14.04)
4.8.4^@^@^@^T^@^@^@^@^@^@^@^@^AzR^@
^A^AESC^L^D^D<88>^A^@^@^\^@^@^@^\^
@^@^@^@^@^@^@4^@^@^@^@A^N^H<85>
^BB^M^Ep<C5>^L^D^D^@^@^\^@^@^@<^@^
@^@4^@^@^@?
```

**3**

```
# 1 "helloworld.cc"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "helloworld.cc"
# 1 "/usr/include/c++/4.8/iostream" 1 3
# 36 "/usr/include/c++/4.8/iostream" 3
```

**4**

```
^@^@^@<A0><85>^D^H^@^@^@^@^R^@^@
^@<FE>^@^@^@p<85>^D^H^@^@^@^@^R^
@^@^@<A4>^@^@^@@<A0>^D^H<8C>^@^@
^@^Q^@^Y^@^@libstdc+
+.so.6^@__gmon_start__^@_Jv_RegisterClasses^
@_ITM_deregisterTMCloneTable^@_ITM_register
TMCloneTable^@_ZSt4endlIcSt11char_traitsIcEER
St13basic_ostreamIT_T0_ES6_^@_ZSt4cout^@_Z
StlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES
5_PKc^@_ZNSt8ios_base4InitC1Ev^@_ZNSt8ios_
base4InitD1Ev^@_ZNSolsEPFRSoS_E^@libc.so.6^
@_IO_stdin_used^@__cxa_atexit^@__libc_start_
main^@GLIBCXX_3.4^@GLIBC_2.0^@GLIBC_2.1
.3
```

Which sequence represents
<output of preprocessor>, <output of compiler>, <output of assembler>, <output of linker>?

a) 1, 2, 3, 4
b) 1, 3, 2, 4
c) 3, 1, 2, 4
d) 2, 1, 3, 4
e) 3, 2, 4, 1

Which lists the components of the compilation process in order from MOST likely to generate an error to LEAST likely to generate an error?

a) Compiler, assembler, preprocessor, linker
b) Compiler, assembler, linker, preprocessor
c) Compiler, linker, preprocessor, assembler
d) Linker, compiler, assembler, preprocessor
e) Preprocessor, compiler, assembler, linker

Which of these is true?

a) A C++ statement like `int myfun(char c, int n);` is called a *function declaration.* It tells the compiler that the full implementation—the *function definition*—is coming later, perhaps from another file.

b) A C++ statement like `int myfun(char c, int n);` is called a *function definition.* It tells the compiler that the full implementation—the *function declaration*—is coming later, perhaps from another file.

c) A C++ statement like `int myfun(char c, int n);` will cause a linker error if there is no full function implementation.

d) Both a and c

e) Both b and c

Consider the following file `things.h`

```
#ifndef THINGS_H
#define THINGS_H

#include<string>

int thing1(int x, int y);
std::string thing2 (std::string x, std::string y);

#endif
```

Which of the following is true?

a) during the `g++` compilation process, the preprocessor notes that the symbol `THINGS_H` has been defined.
b) `things.h` should be passed to `g++` on the command line
c) `things.h` should be `#include`-d in other C++ source files
d) Both a and c
e) Both b and c

Consider the following file `otherthings.h`

```
#include<string>

class Things {
      public:
            int thing1(int x, int y);
            std::string thing2(int x, int y);
};
```

(Something is missing, right?)

And also the file `lotsofwork.h`

```
#ifndef LOTSOFWORK_H
#define LOTSOFWORK_H

#include "otherthings.h"

int foo();

#endif
```

Assume these are accompanied by `.cc` files that define the necessary functions.

Your task: write a `main.cc` that demonstrates the problem caused by missing include guards. Also write a brief description of how this problem will manifest (will the compiler complain? Will the program fail to run? What error message will you see, if any?)

Which of these statements is true?

a) `make` can only execute compiler commands.
b) A `make` rule consists of targets, prerequisites, and recipes.
c) Every `make` target requires an explicit recipe.
d) By default, `make` starts with the target with the most prerequisites.
e) Both b and c.

Write a makefile for the project described below.

The file `shapes.cc` includes definitions relating to drawing primitive graphics on the screen; the publicly useful classes and functions are declared in `shapes.h`.

The file `textures.cc` includes definitions relating to rendering of textures on the screen; the publicly useful classes and functions are declared in `textures.h`.

The file `sprites.cc` includes definitions relating to the animation of graphical “sprites”; the publicly useful classes and functions are declared in `sprites.h`. Some of the classes and functions in `sprites` rely on classes and functions from `shapes` and `textures`.

The main program, `main.cc`, uses `sprites`, `textures`, and `shapes` to display a simple animation.

Modular design of software

a) Has finally become possible with recent innovations in compiler design.
b) Helps to make more maintainable software
c) Makes it easier for teams to write software efficiently
d) Both a and b
e) Both b and c

How should Alice, Bernardo, and Cong divide the work so that they can each be as independent as possible? Remember:

After analyzing the intended behavior of this application, the three decide that they need to implement 5 functions: getint, putint, newline, sum, and product—as well as main.

What do Alice, Bernardo, and Cong need to agree on before they can start working individually?

How can they best use g++ to support their work? Can they run the application when they're working individually? If not, how can they make sure their work is as free from problems as possible?

But wait: if we split up a project using modular design, it seems like we'll have this problem:

a) You have to get everyone's code together before the compiler can find the syntax errors.
b) You have to get everyone's code together before you can run it to see whether there are any bugs.
c) Alice can't use Bernardo's module in her code, because she has no way of knowing what's in Bernardo's module.
d) Cong forgets everything after 10 minutes, so his implementation isn't going to match the interface Alice and Bernardo are expecting him to provide.
e) If Bernardo realizes there's a more efficient way to implement his functions, it's too late, because changing his code might cause problems for Alice and Cong.

What is the output of this code fragment?

```
int x=3, y=4, z=5;
int *ptr1, *ptr2, *ptr3;

ptr1 = &x;
ptr2 = &z;
ptr3 = &y;

*ptr1 = *ptr2;
ptr3 = ptr2;
*ptr2 = *ptr2 + 10;

std::cout
      << std::setw(3) << x
      << std::setw(3) << y
      << std::setw(3) << z
      << std::endl;
```

a) `  3 14  5`
b) `  5  4 15`
c) `  5  4  4`
d) `  5 14  5`
e) `  3 15 15`

What will happen when we try to compile and run this program?

```
 1 #include <iostream>
 2 using namespace std;

 3 int main() {

 4    char c = '?';
 5    int x=10, y=20;

 6    int *ptr1;

 7    ptr1 = &x;
 8    cout << *ptr1 << " ";
 9    ptr1 = ptr1 - 1;
10    cout << *ptr1 << endl;
11 }
```

a) The compiler complains about an illegal assignment on line 7.
b) The compiler complains about an illegal dereference on line 8.
c) The compiler complains about an illegal dereference on line 9.
d) The program compiles, but crashes with a "Segmentation fault." error.
e) The program compiles and runs, but the output is `10 1063089092`

What will happen when we try to compile and run this program?

```
 1 #include <iostream>
 2 using namespace std;

 3 int main() {

 4    char c = '?';
 5    int arr[3] = {4, 5, 6};

 6    int *ptr1;

 7    ptr1 = arr;
 8    cout << *ptr1 << " ";
 9    ptr1 = ptr1 + 1;
10    cout << *ptr1 << endl;
11 }
```

a) The compiler complains an illegal initialization on line 5.
b) The compiler complains about an illegal assignment on line 7.
c) The compiler complains about an illegal dereference on line 10.
d) The program compiles and runs, but the output is `4 7468992846`
e) The program runs with output `4 5`

Assuming we have the following declarations:

```
const int SIZE = 100;
int array[SIZE];
int sum=0, *ptr;
```

**Rewrite** the following code

```
for (int i=0; i < size; i++) {
      sum += array[i];
}
```

with these restrictions:

1. You must use a `while` loop rather a `for` loop
2. You may *not* declare any additional variables.
3. You may *not* use any square brackets `[]`

Determine the output of the following program. Draw and update a diagram of memory to support your work. Write your solution (the output) and your diagram on the board.

```
#include <iostream>
#include <iomanip>

const int SIZE = 10;

int main() {

	int arr[SIZE];
	int *ptr1, *ptr2;

	ptr1 = arr;
	while (ptr1 < arr+SIZE) {
		*ptr1 = ptr1-arr;
		ptr1++;
	}

	ptr1 = arr;
	ptr2 = arr+SIZE-1;
	while (ptr2 > ptr1) {
		if (*ptr2 > *ptr1) {
			*ptr2 = *ptr1;
			*ptr1 = *ptr2;
		}
		ptr1++;
		ptr2--;
	}

	for (int i=0; i<SIZE; i++) {
		std::cout << std::setw(3) << arr[i];
	}
	std::cout << std::endl;
}
```

What's wrong with this implementation of `swap()` ?

```
void swap(int* x, int* y) {

      int *temp;

      *temp = *x;
      *x = *y;
      *y = *temp;
}
```

a) Nothing
b) The compiler will issue an error because `int*` values aren't compatible with `int` values.
c) It will crash because `temp` doesn't point at anything.
d) It will run but it won't swap any values.
e) It will run but it just copies the first value into the second instead of swapping.

The “dynamic” in dynamic memory allocation means that:

a) Programs that have frequently-changing data should store it in special “dynamic memory.”
b) The amount of memory allocated to a program may change from one run to the next.
c) The amount of memory allocated to a program may change during one run.
d) Both a and b
e) Both b and c

What's the problem with this code fragment?

```
 2 int size, *ptr;

 3 std::cout << "How many? ";
 4 std::cin >> size;

 5 if (size > 0) {
 6    int *array = new int[size];
 7    int sum = 0;
 8    for (int i=0; i < size; i++) {
 9          std::cin >> array[i];
10    }
11   ptr = array;
12    for (ptr=array+size-1; ptr > array; ptr--) {
13          std::cout << std::setw(3) << *ptr;
14    }

15    std::cout << std::endl;
16    delete[] array;
17 }
```

a) Nothing.
b) `ptr` points at an "illegal" addresses.
c) The array isn't declared as an array.
d) Dangling pointer.
e) Memory leak.

Deleting line 16

a) Has no effect.
b) Solves the problem.
c) Turns the problem into a memory leak.
d) Causes the program to crash.

The problem with this code

```
int *ptr1 = new int[100];
int *ptr2 = new int[200];
ptr1 = ptr2;
ptr2 = ptr1;
delete[] ptr1;
ptr1 = NULL;
delete[] ptr2;
```

is

a) Memory leak
b) Dangling Pointer
c) Crash
d) All of the above
e) None of the above

Write a function that takes an array of strings and returns an array containing the lengths of each nonempty string (empty strings should be ignored). Its signature is

```
int *getLengths(string[] array, int arraysize);
```

(Remember that you can get the length of a string `s` by saying `s.length()` and you can find out if it's empty with `s.empty()`.)

Code like this

```
3 std::cout << "How many? ";
4 std::cin >> size;
5 if (size > 0) {
6    int *array = new int[size];
```

is great when the user knows ahead of time how much data she's going to give the program. But sometimes no-one knows in advance how much storage is necessary—what if you need storage for each user who joins a chat room? Or for each tweet from the people you follow? These things all happen in real time. What's the best way to deal with this kind of very unpredictable data?

a) Ask the user how much storage to use. If more is needed, just allocate twice as much space and copy everything over.
b) As the programmer, we should just pick a size limit and not even bother to ask the user.
c) As the programmer, just pick a size limit that's way bigger than could ever be needed.
d) Use a data structure that is designed to grow dynamically, just add more space as needed.
e) Use Java; C++ can't handle that kind of unpredictability.

Suppose we were trying to define an abstract data type for the Tic-Tac-Math board. Which of these would *not* belong to such a definition?

a) A Tic-Tac-Math board is a collection of 9 values, each between 1 and 9.
b) Empty cells on the board are represented as zeros.
c) A Tic-Tac-Math board is *winning* if three values from a "row," "column", or "diagonal" sum to 15.
d) Each value can only appear on the board once.

Consider the following abstract data type for a Deck of cards:

> A card must have one of four suits: hearts, diamonds, clubs, spades
> A card must have one of 13 values: two-ten, jack, queen, king, ace.
>
> A Deck *contains* 52 cards
> A Deck can be *shuffled* to put the cards in random order
> You can *deal* the top card in a Deck by removing it

What's the problem:

a) This doesn't include any operations.
b) This doesn't fully specify the values.
c) This doesn't provide any rules of the game.
d) This DOES provide implementation details

Here's the `Circle` class from the book (the version from Figure 7.1):

```
class Circle {
   public:
      double radius = 0.0;

      void setRadius(double r) { radius = r; }
      double getArea() { return 3.14 * pow(radius, 2); }
};
```

Suppose we then declare a `Circle` object:

```
Circle roundThing;
```

Which of the following statements best illustrates the problem with this `Circle` definition?

a) `roundThing.setRadius(47);`
b) `std::cout << roundThing.radius;`
c) `double area = roundThing.getArea();`
d) `roundThing.radius = 0.5;`
e) `delete roundThing;`

The next version of `Circle` declares `radius` as a `private double`. What's the best implementation of `setRadius()`? (Assume any needed header files are `#include`-d)

a)
```
void Circle::setRadius(double r) { radius = r; }
```

b)
```
void Circle::setRadius() {
      std::cout << "Please enter the circle's radius ";
      std::cin >> radius;
      std::cout << std::endl;
}
```

c)
```
bool Circle::setRadius(double r) {
      if (r > 0.0) {
            radius = r;
            return true;
      } else {
            radius = 0;
            return false;
      }
}
```

d)
```
void Circle::setRadius(double r) {
      radius = r;
      std::cout << "Radius set; new area is " << getArea(); << std::endl;
}
```

e) `void Circle::setRadius() { radius = sqrt(getArea()/3.14); }`