Felipe Lima - 109290055

Section 1: c++

1. What is a reference? What is a pointer? How are they different?

A **reference** is an alias (literally a reference) to an existing variable.

A **pointer** is a variable that stores the value of a memory address.

These two are different because a reference when initialized and assigned to a variable it cannot be changed, while pointers can be reassigned. Pointers can also point to a NULL value while references cannot be NULL.

2. Given the following class definition, how would you call the method CalculateMysteries?

```
class Foo {
public:
  Foo(int bar) : bar_(bar) {}

  int CalculateMysteries();

  static int TotalBars();

private:
  int bar_;

  static int baz_;
};
```

```
Create an instance of Foo and call CalculateMysteries using the instance

Foo f(10);
f.CalculateMysteries();
```

3. How would you call the method TotalBars()?

```
TotalBars is a static method, so we can call it two different ways

Using the instance of foo:
Foo f(10);
f.TotalBars();

or using the scope resolution operator by class::function
Foo::TotalBars();
```

4. Can you access the field bar_ from inside the method TotalBars()? Why/why not?

No, bar_ cannot be accessed from inside the methods TotalBars(). This is because in c++ a nonstatic member reference must be relative to a specific object, and you can call TotalBars() without an object of the class.

5. Can you access the field baz_ from inside the method CalculateMysteries()? Why/why not?

No. A non static method is part of an object of the class and a static field span across all objects of that class. So you can't access it in CalculateMysteries().

6. Create a Foo reference, then call one of the Foo methods.

```
Foo f(10);
Foo &ref = f;
ref.CalculateMysteries();
```

7. Create a Foo pointer, then call one of the Foo methods.

```
Foo *ptr = new Foo(10);
ptr->CalculateMysteries();
```

8. Why would you make a field a pointer?

A reason to make a filed a pointer is so that so that a field can be modified in a called function and you still have access to that modified field by the pointer, which is also going to reflect any changes made to the field.

Section 2: Good Coding Practices

1. How would you improve the following code block:

```
bool hasACat = some value from somewhere;

If (hasACat == true) {
  // Do nothing.
} else {
  returnItForADog();
}


Improved:

bool hasACat = some value from somewhere;

If (hasACat == false) {
  returnItForADog();
}
```

2. How would you improve the following code block:

```
int x = some value from somewhere;
int y = some value from somewhere;

if(y > x) {
  cout << "We're messing with numbers!" << endl;
  x += 1;
} else if(x > y) {
  cout << "We're messing with numbers!" << endl;
  y += 1;
} else if(x == y) {
  cout << "We're messing with numbers!" << endl;
  x = x + y;
}
```

Improved:

```
int x = some value from somewhere;
int y = some value from somewhere;


cout << "We're messing with numbers!" << endl;
if(y > x) {
    x += 1;
} else if(x > y) {
    y += 1;
} else {
    x = x + y;
}
```

Section 3: keywords/const/overloading

1. Why would you use a const field? Why would you use a const method?

    Const fields are used to guarantee a variable will not change value throughout the program. A const method is used so the method can't modify the object on which it is called.

2. Where do you instantiate a const field in a class?

    Either in the declaration of the variable or in a initializer list in the constructor of the class.

3. What does inheritance give you?

    Inheritance allows for the definition of a class in terms of another class, that is it allows a class to inherit the members of an existing class.

3. Where and why would you use the "virtual" keyword? What is this concept called and why is it important?

Virtual keyword allows subclasses of the type to override the method. You would use it in front of a method which is present in classes that have a inheritance relationship. It is used when the program needs dynamic dispatch to call the correct function depending on the object used to call it.

This concept is called dynamic dispatch and it means that the binding of the method is determined at run time depending on the type of the object pointed to by the object call. Meaning it calls the respective object that is pointed by the object call - (Most derived version).

4. Why would you need to overload a comparison operator?

A comparison operator needs to be overloaded in the case that you are trying to compare classes or structures (that usually have more than one field) which cannot be compared directly.

Example:

```
struct Point {
  int x;
  int y;

  bool operator==(Point &other) {
    return (x == other.x && y == other.y);
  }
};

p1  == p2;
```

Section 4: Version control & git

1. What is a branch? Why would you work on a non-master branch?

> the code in which all edits affect exclusively that one branch. The master branch is where all the changes will eventually be merged, so it is imperative that the code in this branch works perfectly. This is the reason why users should use their own branch to make edits to the code with the peace of mind that no edits can break the program and when the code is working it can be merged into the master branch. It is a safety and organizational tool.

2. If you are currently working on the branch my-fabulous-feature and your teammate merges another feature into master, what are the commands that you would run to switch to master, get the new code, switch back to your branch, and merge the new code from master into your branch?

> git checkout master
>
> git pull
>
> git checkout my-fabulous-feature
>
> git merge master

3. What is the difference between issuing a "git pull" and submitting a "pull request"?

> Git pull is used to fetch and download the contents of the remote repository into the local one. Pull request is used to merge a branch into the master branch of the repository.

4. When you're reviewing a PR, what are 3 different specific things that you should look for in the code that you're reviewing?

> Does the code compile and run?
>
> What files are being added? (no .o files for exemple)
>
> Style consistency with the code base

Section 5: bash

1. If you are developing in an IDE (such as Sublime, CodeRunner, CodeBlocks, XCode, etc) and your project isn't compiling by pressing the "run/build" button, what are two things that you could do to fix the issue? (Imagine that a friend is having this issue and you are explaining a few things that they could do to figure out what is going on).

Type the commands in the command line to run the file. Exemple:

g++ std=c++11 filename.cpp -o filename

Write a Makefile

Section 6: Unit testing/Catch2

1. What is a TEST_CASE? What is a SECTION? What should each be used for? How many methods should be tested in each TEST_CASE? what about each SECTION?

A TEST_CASE is a macro division in unit testing used to test each function in a program. SECTIONs go inside TEST_CASEs. They are subdivisions used to test specific scenarios and to separate code (code in one section doesn't affect other sections). Each TEST_CASE tests one method and each several SECTIONs test the same method that the TEST_CASE that encloses them tests.

Section 7: Design Patterns

1. For each of the following design patterns, briefly describe what problem they solve **and** how they are implemented in c++.

a. Singleton
    a. Singleton is a design pattern that has a class that only has one single instance.
    b. It solves the problem of when the program requires having only instance of the object of the class. Example: Logger System, messages window in Mac
    c. How:
        i. Private constructor
        ii. static instance field + static GetInstance method
            1. Then get instance by code like Name & n = Name::GetInstance().
        iii. private or deleted copy constructor and assignment operators
b. Flyweight
    a. One instance per type of the given class
    b. Solves: Save memory on apps that you need a lot of objects. Example: games graphics
    c. How: Arrays or maps to store the examples of the objects. Depends on programmer discipline.

c. Prototype
    a. Solve: Creates a copy of the object without creating the actual object
    b. How: provides a Clone() method
d. Factory
    a. Solve: makes objects type flexible. Used to manage the creation of other objects
    b. How: Create a wrapper class or method for object constructors or linked dependencies.
e. Iterator
    a. Solve: a way to access each member of a collection
    b. How:
        i. define the starting point for the iterator
        ii. define how to go to the next element
        iii. define the ending point for the iterator

2. In class, we went over how you will frequently interact with the design pattern Iterator but will rarely implement it yourselves. Why is this?

A lot of iterators are built in libraries.

Section 8: templating/writing generalizable code

1. Write a templated function `void Swap(T & a, T & b)`.

```
template <typename T>
void Swap(T & a, T & b);
```

2. Write a `main()` in which you make at least two calls to `Swap` that do work and two calls to `Swap` that you would expect to work but that don't.

```
Work:                                   Don’t work:
int a = 5;                              Swap(5,7);
int b = 7;
Swap(a, b);

std::string s = "hello";                Swap(a, s);
std::string t = "world!";
Swap(s,t);
```

3. Adjust the non-working function calls so that they do work and write comments about why they did not initially work and what changes you made.

To make the functions work they have to be the same type and have a variable declared first since the parameters of Swap() are references. So Swap(a,b) - variables works for the first one and Swap(s,t) – same type work for the second ones.

4. Why/why shouldn't you write all functions in c++ as templated functions?

Use templated functions when the function can have multiped functionalities and can work with more then one variable type. Do this to avoid writing repeated functions with the only difference being the variable type.

Only use templated functions if you can use the function with more than one variable type. Defining a type for the function is good to avoid error and wrong function calls and arguments.

Section 9: GUIs/Qt

1. What happens behind the scenes for a GUI to respond to a user interaction with the user interface?

There is a connection created either by the programmer of the software connecting a certain action (like pressing a button) to a reaction.

2. In Qt, how many signals can a single object emit? How many slots can respond to a single signal?

An object can emit more then one signal depending on the action that is taken. There can have more then one slots responding to a signal.

3. Give example signatures for a custom signal that has at least one parameter and the slot that you would connect it to. Give an example call to `connect` that you would use to link the to signal to the slot. Describe when you would call `emit` for this signal. You may need to do some research on your own to answer this question!

```
void PlotWindow::on_randomButton_clicked(){}
```

```
connect(ui->addButton, &QAbstractButton::pressed, this , &PlotWindow::SlotTest);
```