PE 12

Name: Oceane Andreis

Section 1: c++

1. What is a reference? What is a pointer? How are they different? A reference is an alias for a variable. A pointer points to a variable in memory. The reference is like a constant pointer and a pointer is like a constant value

2. Given the following class definition, how would you call the method CalculateMysteries? **Instantiate a `Foo` if necessary.**

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

    int CalculateMysteries();

    static int TotalBars();

private:
    int bar_;

    static int baz_;
};
```

Foo one = Foo(10); one CalculateMysteries();

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

Foo::TotalBars();

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

No, because we can call TotalBars on its own without creating an instance of the class which would then lead to not be able to access Foo's private variable.

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

Yes, because baz_ is static

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

Foo &three = Foo(10); three.CalculateMysteries();

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

Foo* Pointer = three; Pointer->CalculateMysteries();

8. Why would you make a field a pointer?

You would want to make a field a pointer elements that interact with other elements like classes or some data structures. You could also make a field a reference so that you could update the reference or the original variable.

Section 2: Good Coding Practices

1. How would you improve the following code block:

```
bool hasACat = true; // this is an example, proceed as if hasACat could be true or false

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

if (!hasACat) { returnItForADog(); }

2. How would you improve the following code block:

```
int x = 3;
int y = 5;

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;
}
```

//new

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

Section 3: keywords/const/overloading

1. **new**: why would you use a const (non-static)…
    1. field/class attribute - You would want a const field if you weren't planning on changing the variable.
    2. parameter - if you don't want the function to change the value of an object or variable
    3. method - A const method would be used if you didn't want to change the instance members
2. Where and why would you use the "virtual" keyword? What is this concept called and why is it important?

You would use virtual so a subclass could access the methods at the lowest level (derived's class method). Virtual will make it sure that the subclass picks not its class function The virtual keyword is used for when a member function of a base class will be overwritten by a child/sub class.

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

When comparing structs or classes without comparison operators implemented.

5. **How do you write the constructor in a derived class? Give an example base class and a class that inherits it. Your example can be similar to, but not the same as, examples from lecture.**

Example:

```
// base class
class Vehicle
{
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};
class Bus: public Vehicle
{  public:
    Bus() // constructor in a derived class
    {
      cout<<"This Vehicle is a bus"<<endl;
    }
};
```

Instance of the class : Bus school_bus = Bus(); Section 4: Version control & git ———- 1. What is a branch? Why would you work on a non-master branch?

A branch depending if it's a local or remote branch is a copy of your master branch. You would want to work on a non-master branch so people can all work on the same file and work on their own part and then later on merge the edits to master.

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 master git checkout my-fabulous-feature git merge master
3. What is the difference between issuing a "git pull" and submitting a "pull request"?

When you do git pull it pulls the latest code from the remote repo. A pull request is requesting that your branch be merged into another branch(for example you have a different branch of master and did some new change to it and you do a pull request to merge with the master branch).

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?
5. No binary files
6. Big library dependencies (in general)
7. Compiled code (no executables, no .o files)

Section 5: bash

1. What is a PS1 in general (what does this control)?

PS1 controls the shell prompt(look)

2. Rate your level of comfort working on the command line on a scale from "not comfortable at all" to "completely comfortable, I am a command line master". Be honest with yourself. What are a few things that you wish you were more comfortable with on the command line?

I am semi comfortable. I can do enough git in the command line that my life is more simple. I can't create folder and text files through the command line. I can edit those text files with vim. I'm not as comfortable with moving and copying things around but that's mainly because it's easier to just drag and drop so I haven't had to do it much.

3. 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).

You could go to the terminal and see what it prints. Or you can use GDB to debug and step through your code. You can also try to comment out portion of your code to localize the issue.

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 function to test a function, it should be used to test large section of a program. A SECTION is a function that can test specific edge case/cases of a function, it should be used to test specific sections. The test case should test each function(all functions) that they are doing what they are supposed to be doing and each section need to make sure the function covers every scenario.

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 class that follows the singleton pattern is a class that only ever has one instance. it can be used for database connections, timers, logging frameworks, thread management classes, and when you need one and only one instance of the class. In c++ you first need to make a constructor private and then you need a static method(getinstance) and a static instance field. You then need to delete or make private the assignment operator & copy the constructor.

~~~~
static Earth& GetInstance() {
        // gets instantiated the first time
        static Earth instance; // guaranteed to be destroyed

        return instance;
    }

  Earth(Earth const&) = delete; // copy constructor
    void operator=(Earth const&) = delete; // assignment operator
```c++
b. Flyweight - http://gameprogrammingpatterns.com/flyweight.html
The flyweight design requires only one instance of each type of a class. You can share certa
You want to use it when individual objects are memory intensive and we are using the same ki
In c++ you want to instantiate one instance per type of your class (e.g. SquareType class)
when you need to reference the wall instance of SquareType (for example), use a pointer
```c++
class Tree
{
private:
~~~~

```
  TreeModel* model_;

  Vector position_;
  double height_;
  double thickness_;
  Color barkTint_;
  Color leafTint_;
};
```

c. Prototype Prototype design would be used when you want to give the client (code) a copy of an object, but not the original object you want more control over how some of the fields are copied.

```
// examples
ClassName * cn = new ClassName();
ClassName * cn2 = new Subclass();

// if we declare like this, we'll get a Subclass, but
// won't be able to pass it to functions that take Creature
// or store in a collection<Creature>
Subclass cn3;


// dynamic dispatch!
ClassName * cloned_subclass = cn2->Clone();
```

d. Factory The factory design patterns is where you have a method/class whose primary purpose is to manage the creation of other objects. Some objects are cumbersome to Create and some object have heavy dependencies. Time & Calendar type objects are a place where you'll commonly see this implemented Time(month, day, year, timezone, hours, minutes, seconds, milliseconds) // lots of parameters. We want to use it when we have objects that are highly configurable and some configurations are common. If we want more control over the creation process we use factory with flyweight. We need to create a wrapper for our constructor.

```
class DiseaseFactory {
public:
static Disease* GetFlu();
};
```

e. Iterator The iterator provides a consistent interface for accessing the elements of a collection in order. The interface loops through a collection of values (you don't have to know how big the collection is or how it's organized) begin, vector::iterator it = vec.begin() next, it++, ++it end/terminate, vec.end() // an example "for each" loop - uses the iterator design pattern under the hood for (type value : collection) { }

// an example of a regular for loop using an iterator rather than an index for (vector::iterator it = vec.begin(), it != vec.end(); it++) {

} You want to use where you implement a data structure that should be traversable.

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?

Iterator are mainly for data structure which would be extra work to implement because most data structures are already implemented for us.

Section 8: GUIs/Qt

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

What happens is the signals and slots interact with each other (whether they are default or custom) to create a user experience. For example when one click a button, the button will do an action that will affect the view.(for example adding a point to a plot)

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

An object can emit as many signals as it wants. You can have as many slots as you want to respond to a single 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.

```
SIGNALS:
   void DeleteMe(classname *p);
private slots:
  void ConnectSlot(classname *p)


connect(p, &classname::DeleteMe, this, &PlotWindow::ConnectSlot);
```

Section 9: templating/writing generalizable code

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

```
template<typename T>

void Swap(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}
```

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

   ```
   struct Point{
    int x;
    int y;
   };
   int main() {
    int a = 10;
    int b = 20;
    double c = 5.0;
    double d = 7.0;
    Point p1 = {0,3};
    Point p2 = {1,5};

    Swap(a, b);
    Swap(a, c); // does not work we can't mix types
    Swap(c, d);
    Swap(p1,p2); //does not work since a struct is a special type of object/variable. We w
   }
   ```

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.

We can't mix types, but if we change our swap function we can make it work for struct. `void Swap(Point p1, Point p2){     Point temp;     temp = p1;     p1 = p2;     p2 = temp; }` 5. Why/why shouldn't you write all functions in c++ as templated functions?

If we call the same function on different types, like ints, doubles, floats, then we would want to template the function so we don't have to create three different functions for similar variables. But, we don't want to template custom types, like classes, enums, or structs, because we can't template these. For these types we need to create our own functions, so attempting to create a template is more work than it needs to be.