Name: David Skrenta

Turn this file in on canvas before the end of class.

This review is meant to be a check-in with yourself. Which of the topics are you confident about? Which are you less confident about? What do you need to review?

For each question, answer without looking at the internet, then compare answers with your table-mates and your notes.

Section 1: c++
----
1. What is a reference? What is a pointer? How are they different?

A reference is a memory address, a pointer can point to any memory address or 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_;
};
```

```
Foo demoFoo = Foo(12);
demoFoo.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?

You cannot because bar_ is private and TotalBars is static.

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

Yes, you can access the static variable Foo::baz_ because it is available anywhere the class exists.

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

```
Foo& fooRef = Foo(12);
fooRef.CalculateMysteries();
```

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

```
Foo* fooPointer = fooRef;
fooPointer->CalculateMysteries();
```

8. Why would you make a field a pointer?

You would make a field a pointer if you wish to not have that value copied.

Section 2: Good Coding Practices
----

1. How would you improve the following code block:

```
bool hasACat = true;

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

```
if (!hasACat) {
  returnItForADog();
}
```

or

```
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;
}

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

Section 3: keywords/const/overloading
----

1. Why would you use a const (non-static)...
   i. field/class attribute
   ii. parameter
   iii. method

i. When the field/class attribute does not change ever.
ii. Use const when the parameter cannot be modified.
iii. Const functions cannot modify the objects they are called on

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

The virtual keyword is used for when a member function of a base class will be overwritten by a child class.

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

When comparing structs or classes without comparison operators implemented.

5. What is one thing about inheritance that you wish that you knew better?

I wish I had more examples where it was clear that inheritance was a better way of organizing it than other methods.

Section 4: Version control & git
----
1. What is a branch? Why would you work on a non-master branch?

A branch is a version of the codebase. You would want to work on a non-master branch so others can work on the same codebase at the same time without merging difficulty.

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 pulls the latest code from the remote while a pull request is requesting that your branch be merged into another 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?

1. Style/Conventions
2. Compilation
3. No binary files

Section 5: bash
----
1. What is a PS1 in general (what does this control)?

PS1 is the variable that holds your shell prompt.

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 feel I am mostly comfortable with the command line.

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).

Use the command line to remove abstractions and hidden details.

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 desegined to test another function and verify its return value. A section is a group of test cases designed to completely test a function. One method should be tested in each test_case and section.

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 single class exists, each "new" instance only returns the single instance.

b. Flyweight

Allows you to share certain parts of the class across multiple instances in order to save space.

c. Prototype

Prototype instance is cloned to create new instances.

d. Factory

Objects can be created without a close connection to the class structure of the library.

e. Iterator

Each collection will provide an iterator that allows you to iterate through the objects without exposing the implementation detains of those underlying classes.

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 is frequently used in public libraries.

Section 7: Design Patterns
----

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

A signal is generated by the user interaction and a slot acts as a listener in order to trigger an event.

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

An object can emit a single signal, multiple slots can respond 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.

```
void CustomSignal(int num);

class SomeClass : QObject {
  Q_Object
  int internval_value;

  public:
    int get_value() { return internal_value; }

  public slots:
    void updateValue(int value);

  signals:
    void onValueUpdated(int updatedValue);
}
```

Slot implementation:

```
void SomeClass::updateValue(int value) {
  internal_value = value;
  emit SomeClass::onValueUpdated(value);
}
```

Connect:

```
SomeClass foo1;
SomeClass foo2;
QObject()::connect(&foo1, SIGNAL(onValueUpdated), &foo2, SLOT(updateValue(int)));
```

Section 9: templating/writing generalizable code
----

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

```
template <class 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.

```
int main() {
  char a = 'a';
  char b = 'b';
  Swap(a, b);
  std::cout << a << ", " << b << std::endl;

  int one = 1;
  int two = 2;
  Swap(one, two);
  std::cout << one << ", " << two << std::endl;

  std::string str1 = "str1";
  std::string str2 = "str2";
  Swap(str1, str2);
  std::cout << str1 << ", " << str2 << std::endl;
}
```

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.

All types I tested worked with the template Swap function implemented above. I tested classes, structs, and pointers to both.

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

Templates increase compilation time, templates leak implementation details in the header file, templates further obfuscate error messages and can decrease code readability.