Review Topics

Make sure to also review the **Exam 1 Study Guide** to review topics that have been previously covered. Exam 2 contains questions on topics from Exam 1.

Loops

Be familiar with writing for-loops, both forward and backward.

```
for ( int i = 0; i < 10; i++ )
{
    cout << i;
}

for ( int j = 100; j > 0; j-- )
{
    cout << j;
}
```

Arrays

Initializer List

You can initialize an array with an initializer list.

```
string names[] = { "Yakko", "Wakko", "Dot" };
```

Must know size at compile-time

For these types of arrays, the size of the array must be known at compile time. Therefore, you either need to declare an array with a size (like in 24), or immediately initialize it with an initializer list (like in 25).

Track max size and current size

An array won't automatically track its maximum size or the current amount of objects it contains. You will need to keep track of these yourself, usually with variables.

```
string students[30];
int maxStudents = 30;
int currentStudents = 0;

students[ currentStudents ] = "Bob";
currentStudents++;
```

Passed by reference by default

Arrays, by default, are passed-by-reference in C++, because arrays may be large, and very costly to copy over. You can make sure values do not change in the array by passing it as a const.

```
void DisplayList( const string names[], int size );
```

Functions

Be familiar with terminology related to functions, as well as how to declare, define, and call functions.

Terminology

Function Declaration
Ends with a semi-colon – no function body.

```
int Sum( int a, int b );
```

Function Definition
Defines the functionality – includes function body

```
int Sum( int a, int b )
{
    return a + b;
}
```

Function Body
The part of a function within the curly braces

Function Header or Function Signature
The part of a function above the curly braces – the return type, name, and parameter list

Return type
The return data-type of the function

```
int Sum( int a, int b );
```

Function name
The name of the function. Should be descriptive.

```
int Sum( int a, int b );
```

Parameter list
The list of variables that must be passed into the function from the caller.

```
int Sum( int a, int b );
```

Function call
Where the function is being called from, and will return to.

```
int total = Sum( 1, 3 );
```

Declaration vs. Definition

The declaration is the function header with a semi-colon at the end. Can be declared above main, so that it can be defined below main.

The definition contains the actual inner code (the function body).

Parameter vs. Argument

The parameters are the variables that the function expects to be passed in. These are in the function header, in the parameter list.

The arguments are the variables that are *passed into* a function call. These are outside of the function definition, and are whatever variables or values are passed in on call.

Parameters a, b
```
int Sum( int a, int b );
```

Arguments 1, 3
```
int total = Sum( 1, 3 );
```

Return types

A function can return any data-type, but also **void**. The void return-type is used when the function does not return any values.

Pass by value vs. Pass by reference

All variables, besides arrays, are passed by value by default. This means that the *argument* variable is *copied*, and the function works with a copy. If any changes are made to the copy, it does not affect the original argument variable after the function ends.

A variable can be passed by reference by adding on the & operator after the data-type. If a variable is passed by reference, its value can change within the function. You can use this technique to return multiple values, as you can only return one item via the `return` statement.

New Topics

Structs and Classes

How to declare a struct and class, its member variables and functions.

```
struct CoordPair
{
    float x, y;
};

class Student
{
    public:
```

```
    void Setup( const string& name, float gpa );

    private:
    string m_name;
    float m_gpa;
};
```

What **public**, **protected**, and **private** mean.

- Public members are accessible outside of the class. If a member variable or function is public, it can be accessed directly through the instantiated object.

```
Student student;
student.name = "Bob";              // public member variable
float gpa = student.GetGPA();      // public member function
```

- Private members are only accessible to the class it belongs to. Private members are not available to any children of the class.
- Protected is utilized for a class that will have children. Protected members are not available outside of the family tree.

Creating separate .cpp and .hpp files for class objects

It is standard to keep your class and function declarations within a header (.hpp or .h) file, with the function definitions within a source (.cpp) file.

Within any header file, you need to wrap it with the following:

```
#ifndef _MYCLASS
#define _MYCLASS
// Code goes here
#endif
```

This ensures that the compiler does not try to “copy” the code from this file multiple times, which causes a redeclaration error.

When defining functions in a .cpp file and outside of the class declaration, you need to specify what class it belongs to with the scope resolution operator ::

```
void Student::SetGpa( float gpa )
```

```
{
    m_gpa = gpa;
}
```

Constructors

Class constructors are a special type of function in a class. The constructor must have the same name as the class. Constructors are called immediately upon instantiation of an object.

Overloading Constructors

Like with function overloading, you can declare multiple constructor functions that have different parameter lists. This can come in handy if you want to initialize a class, but may have different amounts of information available when the object is instantiated.

Destructors

Destructors are called automatically once an instantiated object loses scope – when the object is destroyed. You cannot overload a destructor. The destructor must share the name of the Class, but with a tilde ~ at the beginning. No parameters.

Struct vs. Class

Keep in mind that with members of Structs, they are public by default. With classes, members are private by default. Structs and Classes can both have member variables and functions.

Inheritance

How to inherit

You can create a class that inherits any public and protected members from another class:

```
class Student : public Human
{
    // ...
};
```

```
class Teacher : public Human
{
    // ...
};
```

Calling parent constructor and functions

Within a child class constructor, you can explicitly call the parent constructor through the initializer list. You can pass in any arguments that are needed.

```
Student::Student() : Human()
{
    // ...
}
```

Overwriting / Redefining

Child classes can overwrite (aka redefine) functions that were originally declared in the parent class. If the parameter list is the same, you are overwriting that original function. Within the new version of the function, you can also call the parent version as needed.

```
void Student::AddZipcode( int zip )
{
    Human::AddZipcode( zip );
}
```