C++ Basics

1. Declaring a variable and initializing variables

Know that, without initializing a variable, it will contain garbage.

```
int number;
int count = 2;
float cost = 2.39;
char letter = 'A';
string word = "Hello!";
```

2. Data Types, and values they can contain

Integer Whole numbers	**Float, Double** Numbers that may have a decimal point	**Boolean** True or False. Can be assigned int; 0 = false, any other number = true.
String Values are stored within double-quotes.	**Character** Single character. Values are stored within single-quotes.	

3. Input and Output

Remember that << and >> are called **stream operators**. Remember which direction they point for each item. Also note that you can chain together string literals (within double-quotes) and variable values.

```
cin >> amount;
cout << "Amount: " << amount << endl;
cout << "\n New Line" << endl;
cout << "\t Tab" << endl;
```

4. Types of Errors

Syntax Error	**Runtime Error**	**Logic Error**
Syntax errors are due to typos or misspellings, and violate the rules of the C++ language. The compiler will catch these.	A runtime error is not caught by the compiler. The error occurs when the program hits bad code, and may cause a crash.	A logic error won't necessarily cause a program error (it might eventually). Logic errors could be incorrect formulas, or incorrect program flow.

Control Flow

5. Boolean Expressions

Be familiar with types of boolean expressions. Realize that all of these are questions that return either true or false, which can be used with if statements and loops.

Is equal to	**Is not equal to**	**Less than**
var1 == var2	var1 != var2	var1 < var2
Greater than	**Greater than or equal to**	**Less than or equal to**
var1 > var2	var1 >= var2	var1 <= var2
Is true	**Is false**	
var1 == true var1 != false var1	var1 == false var1 != true !var	

Note that you can check if something is true or false with the shorthand:

```
if ( var1 ) // if true
if ( !var1 ) // if false
```

6. And, Or

You can chain together boolean expressions in a single If Statement or Loop to check for multiple conditions.

```
if ( var1 < 0 || var1 > 10 ) // between 0 and 10
if ( var1 >= 0 && var1 <= 10 ) // between 0 and 10
if ( var1 >= 0 && var1 <= 10 )
```

7. How to write an If Statement, Else If, Else

The if statement will stop checking additional conditions (the other if / else ifs) once it finds a match. The else statement is executed if all prior checks were all false.

```
if      ( choice == 'a' ) { /* ... */ }
else if ( choice == 'b' ) { /* ... */ }
else if ( choice == 'c' ) { /* ... */ }
else                      { /* ... */ }

if ( var1 < 0 ) { cout << "var less than 0"; }
else { cout << "var greater than or equal to 0"; }
```

8. Break and Continue

You can use a **break** statement to break out of a loop. You can use **continue** to skip the rest of the code within a loop, but continue the loop, such as if there is an error.

```
while ( true )
{
    cin >> choice;
    if ( choice == 'q' ) { break; }
}
```

```
while ( true )
{
    cin >> choice;
    if ( choice == 'a' ) { continue; }
    cout << "Hello" << endl;
}
```

9. How to write a While Loop

Think of a while loop like an if statement, except it will keep looping until the condition is false. You can also use `&&` and `||` to add additional conditions.

```
int count = 0;
while ( count < 100 )
{
    count++;
}
```

10. How to write a Do-While Loop

With do-while loops, the inner contents will always be run *at least once*. Then, if the condition is false, the loop will stop. Otherwise, if the condition is true, it will keep looping.

```
do {
    cout << "Enter a #: ";
    cin >> choice;
} while ( choice < 0 || choice > 10 );
```

11. How to write a Switch Statement

You can use a switch statement to check the value of an integer or character variable. Be aware that each case statement needs to end with a break; - if it does not, it the program will execute the case statement directly afterwards as well. default: is used like the “else” statement.

If choice = 'b', the output will be BC , since the 'b' case does not have a break;

```
switch( choice )
{
    case 'a':
    cout << "A";
    break;

    case 'b':
    cout << "B";

    case 'c':
    cout << "C";
    break;

    default:
    cout << "D";
}
```

12. How to write a For Loop

For Loops are good for counting. Note that the first part is creating and/or initializing a variable, the second part is the condition, and the third part is a command that will be done each time through the loop.

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

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

13. Nested Statements

Once inside a loop or an if statement, you can continue adding if statements and loops within.

A nested if statement can also be written with a single if statement and `&&`.

```
if ( var1 > 0 )
{
  if ( var < 10 )
  {
    cout << "0 < x < 10”;
  }
}
```

=

```
if ( var1 > 0 && var1 < 10 )
{
  cout << "0 < x < 10";
}
```

Functions

14. 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 );
```

15. 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).

16. 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 );
```

17. 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.

18. 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.

```
void InitVars( int& a, char& b ); // Pass by reference
```

19. Overloading Functions

Function names can be reused, so long as the function signature is unique. To have a unique function signature, there needs to be either a different amount of parameters, or different data-types for parameters. Then C++ can tell the functions apart, and will choose the correct one based on the data types of the arguments that are passed in.

Overloaded Functions

```
void Display( int a, int b );
void Display( char a, char b );
void Display( string a, string b );
```

Ambiguous – will not compile

```
void Display( int a, int b );
void Display( int x, int y );
```

20. Scope

Usually, you can tell a variable's scope by the curly braces it is contained within. (The code within curly braces is known as a code-block.) Once the program leaves a code block, the variable goes out of scope, and is destroyed. Therefore, if you declare a variable within an if statement, you cannot use that variable outside of the if statement.

This also applies to functions. Different functions can reuse the same variable name, since they are not in the same scope.

21. Default Parameters

You can specify default values for parameters in a function's parameter list, which are set if the calling function does not provide a value for that item.

```
// Default value of 4.0 if a GPA value isn't passed in
void Display( string name, float gpa = 0.0 );
```

```
Display( "Makoto" );   // Uses default 0.0
Display( "Ami", 4.0 ); // Overwrites with 4.0
```

Const

22. Const Variables

You can declare a variable as const, so that you don't have to hard-code some value all over the place. This makes it easy to change the single value, rather than hunting through the code for where the number has been hard-coded.

A const variable must immediately be initialized to some value. It is best-practice to name const variables in ALL CAPS.

```
const int MAX_STUDENTS = 30;
```

23. Const Parameters

If you want to pass an object as an argument by-reference, but do not want the value of that object or the object's members to change, you can pass it as a const reference. Then, you receive the benefit of not passing-by-value (not copying the variable), but still ensure that the variable doesn't change.

```
void Display( const string& name );
```

Arrays

24. Declaring 1D, 2D, etc.

```
string question[5];
string answers[5][4];
```

25. Initializer List

You can initialize an array with an initializer list.

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

26. 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).

27. 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++;
```

28. 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 );
```