C++
Glossary

#include - The **#include** command tells the computer to include a library or a form. Your computer will understand some commands automatically, but if you want to use a command that is in a library, you need to tell the computer to include the library, first. When you include forms or other files, you use quotes around the name of the form instead of < and >.

(Name) - In Visual C++, the **(Name)** property tells the computer what your object's name is. Not all objects have this property. When programmers need to write code that uses an object, they tell the computer which object they mean by using the value stored in the (Name) property.

The (Name) property can't be named Name. Visual C++ doesn't allow you to name an object Name because the code would get confusing.

+ - The **+** operator adds two numbers together.

- - The **-** operator subtracts one number from another.

*** -** The ***** operator multiplies two numbers together.

/ - The **/** operator divides one number by another.

% - The **%** operator, or modulo operator, divides one number by another and figures out the remainder.

For example, if you divide 5 apples between two people, each person gets 2 apples and there is 1 apple left over. The leftover apple is the remainder.

Computer programmers sometimes use the % operator to limit a random number.

For example: `Number = rand ( ) % 10;`

The computer divides a random number by 10, and then stores the remainder in Number.

In this example, the remainder could be any number from 0 to 9. No other remainder is possible when you divide by 10.

++ - See **Increment and Decrement**.

-- - See **Increment and Decrement**.

== - See **Comparison Operators**.

<< - The **<<** operator is used when you output information. The << arrows point **out** to where the information is going.

>> - The **>>** operator is used when you input information. The >> arrows point **in** to where the information is being stored, usually to a variable.

&& - The **&&** (and) conditional operator tells the computer to check two conditions, one on each side of the &&.

```
if (Apples == 2 && Oranges == 2)
{
    shareFruit ( );
}
```

In the example, the computer will run shareFruit ( ) only if the value of Apples is 2 **and** the value of Oranges is 2.

!= - The **!=** conditional operator checks to see if the value on the left is not equal to the value on the right. For example:

```
while (Guess != Secret_Number)
{
    // This code repeats while the guess
    // is not equal to the secret number
}
```

-> - The **->** operator is a way of telling the computer which object's methods you want to use.

The -> operator can also point to an object within an object. For example:

```
Car_1 -> Engine
```

The Engine object in the code above is the engine that belongs to Car_1.

^ - See **Data Management**.

.cpp Files - .cpp is the file extension for a C++ file. The cpp stands for C Plus Plus. If a file has a .cpp on the end of its name, that means it's a C++ file.

.exe Files – See **Executable Program**.

Array - An **array** can store many values in one variable. All of the variables you learned about so far can be turned into arrays. Variables that become arrays have more than one spot to store information in. For example, you could store the numbers 1, 52, -24, 0, and 9 in just one int array with 5 spots for information.

Array declarations are like variable declarations except they use straight brackets, like this:

```
char Board[9];
```

The straight brackets tell the computer that Board[ ] is an array. The number between the brackets is the size of the array. In this example, there are 9 spots to store a char in the Board variable.

Arrays are numbered in an unusual way. The number of things in an array starts with 0 (zero) and ends with one less than the number in straight brackets.

Example: `char Board[9];`

In the example above, Board[9] is an array with 9 spots for information. The nine spots are: Board[0], Board[1], Board[2], Board[3], Board[4], Board[5], Board[6], Board[7], Board[8]

When you assign a value to an array, the computer needs to know what spot to put the value in.

An ordinary char might store a value this way: `Board = 'x';`

A char array would store a value like this: `Board[2] = 'x';`

In the second example, the computer stores the letter 'x' in the 2 spot of the Board array.

Attribute - In XML, an **attribute** provides more information about an element. Attributes are written inside the opening tag of an element. Attributes can have values, just like variables. In fact, attributes look a lot like variables that are being assigned a value.

Auto Formatting - Auto formatting is when the computer changes the color of the words that you type, or even moves your words around on the screen. This doesn't mean your computer is having problems—it's to help you create your programs by making your code easier to read.

bool - A **bool** variable can only have two values: true or false. You declare a bool variable the same way as other variable types: `bool You_Are_Smart = true;`

Brackets – There are curly brackets, like { and }. There are also straight brackets, too, like [and]. If you use the wrong kind of bracket, your program won't work, so pay attention to which kind of bracket you need.

C++ programs won't run unless there are the same number of opening brackets as closing brackets. For every { there has to be a }. To help keep track of how

many of each type of bracket there are, computer programmers use indentation. See **Indenting Code**.

break - The **break** command tells the computer to stop running code inside the current statement. Computer programmers use the break command to avoid running code they don't want to run.

Bug - A **bug** is an error in your code. Common examples of bugs include forgetting a semicolon at the end of a line, or leaving out a closing bracket. If your program has a bug, it probably won't run.

Button - A **button** is a part of the interface that a person can click with the mouse to interact with a program. A button usually triggers an event, like starting a search or opening a dialog box. Buttons are also used to confirm an action. Buttons are usually rectangles that are wider than they are tall, but they can be other shapes, too. See **Event**.

Calling a Function - Computer programmers **call** a function to tell the computer to run the code inside the function's curly brackets.

char - A **char** (character) variable can store one character of information. For example, a char variable could store one digit numbers (like '5') capital letters (like 'A') or lowercase letters (like 'd'). A char variable can't store negative numbers (like -3), big numbers (like 25), or more than one letter (like cat). A char variable's value is always in single quotes.

cin - The **cin** command gets input from the user and gives it to the computer.

Class - A **class** describes what variables and functions an object will have.

When you create an object, it automatically has the variables and functions described by the class. After it's created, you can change values stored in its variables to make it unique.

You can create more than one object from the same class. All objects from the same class are able to store the same information and use the same functions.

Class Definition - To create a class, define the class with the variables and functions you want the class to have.

```
class Name_Of_The_Class
{
    // The variables and functions of the class go here.
};
```

A class definition looks like the example. The variables and functions for the class go between the curly brackets.

You don't usually put a semicolon after curly brackets, but since this is a class definition, it needs one, just like a variable declaration or function declaration does.

Code - Code is a collection of commands and instructions that tell a computer what to do.

Command - A **command** is an instruction you give the computer.

Comment - A **comment** is a part of the code that the computer ignores. Programmers insert comments into computer programs to explain their code to themselves and other people. Computer programmers put comments before each command or group of commands.

In C++, comments start with two slashes, like this:

```
// This is what a comment looks like.
```

When a computer sees the two slashes, it ignores whatever comes after them on that line. That way, you can write notes to yourself or other people without messing up the program.

A /* */ comment is a comment with line breaks in it, so it takes up more than one line. The /* begins the comment, and the */ ends the comment.

Some programmers use /* and */ to comment out code that might have an error, so they can try running the program without that part of the code.

The /* and */ are also useful for making multi-line comments that introduce your project.

When computer programmers want to explain something about a particular line, they comment on the end of that same line. Comments at the end of a line of code use the // to tell the computer to ignore the rest of the line.

Comparison Operators - A **comparison operator** tells the computer to compare two things. ==, >, and < are just a few comparison operators.

The **==** tells the computer to see if what's on the left side is the same as what's on the right side. It compares the things on each side of the == to see if they are equal.

> is the greater than operator. It checks to see if the number on the left is higher than the number on the right.

< is the less than operator. It checks to see if the number on the left is lower than the number on the right.

Compiler - A **compiler** is a software program that changes code into machine code.

Compiling - Compiling is the process of changing C++ code into machine code. The computer uses the compiled machine code to run your program.

Computer Programmer - A **computer programmer** is someone who writes computer programs. They write code to tell a computer what to do.

Conditional Statement - A **conditional** statement tells the computer to do something if a condition is true.

In an if ( ) statement, the condition is between the (and the). For example:

```
if ( condition )
{
      // These commands run if the condition is true
}
```

Console Project - A **console project** is a type of C++ project that opens a black window and uses text to communicate with the user.

cout - The **cout** command is for output. The cout command can be used to print variable values to the screen.

Data Grid - A **data grid** is a list of data with columns and rows. In Visual C++, you can create a data grid using the DataGridView object.

DataGridView - A **DataGridView** object has rows and columns built into it. Once a program has a data grid, you can add columns to it.

Data Managed Object - See **Data Management**.

Data Management - Data management is a computer's way of keeping track of how a program is using its memory. A **data managed object** helps the computer manage its memory by getting rid of data when it's no longer needed. Complex objects like the DataSet object need to be data managed. The **^** symbol tells the computer that an object should be data managed, which helps your program run more smoothly.

DataSet - A **DataSet** object holds information in the computer's memory. A DataSet object is particularly good at storing information in columns and rows.

Debugging - Debugging is the process of finding and fixing bugs in a computer program. If your code has bugs, the program won't run. That's okay, though,

because debugging is a normal part of being a computer programmer. It's okay to have bugs!

Decrement – See **Increment and Decrement**.

Default Buttons - Default buttons are buttons with built-in behaviors. For example, AcceptButton is a common default button. When you see a form with an AcceptButton, that button is automatically highlighted when the form is opened. In a dialog box, you might see buttons that say OK or Cancel. The OK button has been set as an AcceptButton, and the Cancel button is a CancelButton.

do while ( ) - A **do while ( )** loop is almost the same as a while ( ) loop, but it runs code inside the brackets at least once before checking the condition.

```
do
{
    // This code runs once for sure. Then, it keeps running until
    // the condition isn't true.
}
while (Condition == true);
```

The while in a do while ( ) loop comes after the curly brackets.

Element - See **XML Element**.

else - An **else** statement tells the computer what to do if an if ( ) statement doesn't run.

```
if (This_Thing == 1)
{
    // Do this if This_Thing equals 1
}
else
{
    // Otherwise, do this
}
```

An else statement comes right after the } in an if ( ) statement, like in the example.

In the example, if This_Thing equals 1, the computer ignores the else statement.

If This_Thing doesn't equal 1, the computer skips the if ( ) statement and runs the else statement.

else if ( ) - The **else if ( )** statement is an if ( ) statement attached to an else statement.

The else if ( ) statement always follows an if ( ) statement. It will only run if the first if ( ) statement is false and the else if ( ) statement's condition is true.

```
if (Condition 1)
{
    // Run this code if Condition 1 is true
}
else if (Condition 2)
{
    // Run this code if Condition 1 is false
    // and if Condition 2 is true
}
```

endl - The **endl** command tells the computer to move on to the next line when it's printing text on the screen. If you don't end your lines with endl, then all of your text will end up on one line. If it's all on one line, it's harder to read.

Event - An **event** is an action that is caused by the user or another part of the computer, such as the operating system. For example, an event happens when a user clicks a button or presses a key on the keyboard.

Event Handler - An **event handler** detects when an event happens and runs code written only for that event. For example, when a button is clicked, an event handler for that button runs that button's event handler code. In Visual C++, when you double-click certain kinds of objects, like a button, an event handler section of code is automatically created for that object.

Executable Program - An **.exe** is an **executable program**, a program that can be run without any other software.

Fields - Sometimes data can be divided into smaller parts called fields. A **field** is a small piece of a larger amount of data. For example, items in an organizer might have two fields of data, such as Item_Name and Category.

float - A **float** variable is a number variable that can have a decimal point (a "floating point"). A float variable could store the number 5, or the number 3.45, or even a negative number, like -6.78.

Flowchart - A **flowchart** is a picture of a process.

for ( ) - A **for ( )** loop is a loop that's designed to do something a certain number of times.

For example, this for ( ) loop will count to 10.

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

fstream - The **fstream** library contains commands for reading and writing to files.

Function - A **function** is a group of statements and commands. Programmers group code into functions to organize their code and make programming easier. Once a function is written, it can be used again and again without writing out all of the code each time.

Functions have curly brackets { } just like the main ( ) function. When the computer runs a function, it runs the code inside these curly brackets. Computer programmers usually put functions at the end of their program, after the last } of the main ( ) function.

Function Declaration - A **function declaration**, or **prototype**, is a way of telling the computer you'll use a function later in your program. Prototypes look a lot like variable declarations with parentheses ( ) added to them.

Global Variable - A **global variable** can be used by any function. A global variable is declared outside of all functions, before the main ( ) function.

if ( ) - An **if ( )** statement tells the computer to do something only if a condition is true. For example:

```
if ( condition )
{
    // The computer will run the code inside these brackets if the
    // condition above is true.
}
```

include - See **#include**.

Increment and Decrement (++ and --) - The **++** (increment) operator adds 1 to an int variable. The **--** (decrement) operator subtracts one from an int variable. For example:

```
int Total_Moves = 4;
Total_Moves++;
```

In the example above, the value of Total_Moves would be 5 after all of the code is run, because the ++ operator adds 1 to the original value of 4.

The -- (decrement) operator can be used the same way, except it subtracts 1 instead of adding.

Indenting Code - Computer programmers indent code to make their programs easier to understand. The following guidelines can help keep your code organized:

- Indent the opening bracket so it's under the first letter of the statement that it belongs to.
- Indent the code within that bracket.
- Indent the closing bracket the same amount as the opening bracket.

Input - Input is a word for information that you send to the computer.

Instance – See **Object**.

int - An **int** variable is a number variable that can't have a decimal point. An int variable could store the number 5, or the number 32, or even a negative number, like -23, but not numbers with decimal points, like 2.1 or -3.45.

Integer - Integers are numbers without decimal points.

iostream Library - The **iostream** library has commands in it that involve input and output. You'll learn more about input and output later.

Label - A **label** is text that tells the user what something is for. Labels help the user understand how to use the program, usually by labeling parts of the program that need a short description.

Library - A **library** is a collection of commands that can be used for a specific purpose. The libraries used in this course are built into C++, but some programmers create their own libraries, or use libraries created by other programmers.

Local Variable - A **local variable** can be used only by the function it's in.

Loop - A **loop** is a section of code that the computer runs over and over. Computers will keep repeating a loop until a condition is met.

Machine Code - Machine code is code that can be read by a computer.

The C++ language was designed to be read by humans. For a program to run, your C++ code has to be converted into machine code.

main ( ) Function - The **main ( )** function of your program is the part that the computer runs. This is sometimes called the main ( ) loop. To run a C++ console program, a computer runs the commands within the main ( ) function. If there isn't a main ( ) function, the program won't work.

All the commands in a main ( ) function go between an open and closing bracket, like:

```
{
    // Commands go here.
}
```

Menu Strip - A **Menu strip** is the bar at the top of a window with one or more menus on it. In Visual C++, you can create a menu strip using the MenuStrip object.

MenuStrip - A **MenuStrip** object creates a menu bar at the top of a form. Once a program has a menu strip, you can add menus to it, like File, or View.

Message Boxes - A **message box** is a part of the Windows interface. It is a box that appears with a message for the user.

Method - A function that belongs to a class is called a **method**. Methods are also called **member functions**.

Like other functions, methods need to be declared, written, and called. Look at the example to see how a method called methodName ( ) might be declared inside a class.

```
// Declare the class
class Class_Name
{
private:
    string Variable_Name;
public:
    void methodName ( );
};
```

Name – See **(Name)**.

Namespace - A **namespace** is a way to group commands. The namespace you use for your program affects the way you name your commands.

For example, without using a namespace, you would write a cout command like this:

```
std::cout
```

If you use the standard (std) namespace, you can write the cout command in a much easier way, like this:

```
cout
```

Object - An **object** is a special type of data that has its own variables and functions. An object that's part of a class is often called an **instance** of a class. More than one object can be created from a class.

Object-Oriented Programming - Object-oriented programming is a type of computer programming that uses objects. See **Object**.

Output - Output is information that a computer sends to the user.

Pause – See **system ("PAUSE");**.

Private - Classes have private and public sections to keep their variables and functions in. **Private** variables and functions can only be used by the object or its functions. Programmers make variables and functions private to control how those variables are used by other parts of the program.

Prototype – See **Function Declaration**.

Public - Classes have private and public sections to keep their variables and functions in. **Public** variables and functions in a class can be used by other parts of the program. They are the opposite of private because they aren't restricted to being used only by the object and its functions. Programmers use public functions to change values stored in private variables. That way, they control how other parts of the program use the variables that are private.

rand ( ) - The **rand ( )** function creates a number that is pretty close to random.

The rand ( ) function works by keeping a list of unrelated numbers. When the rand ( ) function runs, it grabs the next number in the list. In most cases, rand ( ) is close enough to random that it can be used for creating random numbers. See **srand ( )**.

Random Number - A **random number** is a number chosen without using a system. Random numbers can't be predicted. Unfortunately, computers can't choose random numbers because they use systems to do everything.

Reading from Files - A computer **reads** information from a file by getting information from the file. See **Writing to Files**.

Remainder - The **remainder** is the amount left over when one number is divided by another.

return - The **return** command tells the computer what value a function is giving back. The return command also ends the function. Once the computer runs the return command, it stops running code in that function. This is similar to how the break command works. See **break**.

ShowDialog ( ) - The **ShowDialog ( )** method tells a form to appear. It's what makes a dialog box show up when you click a button or menu option.

srand ( ) - The **srand ( )** function chooses a spot in the rand ( ) function's list of numbers to start at.

This can be done many ways, but this is a common example:

```
srand (time (NULL) );
```

The srand ( ) function chooses a spot from the list based on the exact time of day. This is called seeding the rand ( ) function.

Stream - In C++, a **stream** is information that travels to or from a computer program.

Before you use a stream, you have to create one. Creating a stream is just like creating an object. In fact, streams are a special type of object that C++ programmers can use.

After you create a stream, you still need to open it. When you're done with a stream, you need to close it. Anything you want to use the stream for has to happen in code that comes between the opening and closing of the stream.

string - A **string** is a line of text. A string variable stores a line of text. A string variable is very flexible. It can store values like, "Peanut butter and jelly is my favorite." String values are always in double-quote marks, and computers need the string library to know how to use them.

switch case - A **switch case** statement decides what to do based on the value of a variable.

```
switch (Apples)
{
case (0):
    {
      goHungry ( );
    }
case (1):
    {
      eatApple ( );
    }
case (2):
    {
      inviteFriend ( );
      eatApple ( );
    }
}
```

In the example, the computer gets the value of Apples. If Apples is 0, the computer runs the code in case (0). If Apples is 1, the computer runs case (1). If Apples is 2, the computer runs case (2).

You can add as many cases as you want to a switch case statement.

system ("PAUSE"); - The **system ("PAUSE");** command pauses the program. When this command is run, the program waits for the user to press any key to continue.

Text Box - A **text box** is an area of a program where the user can type text. A text box looks like a box. Users enter text in them by clicking the text box, then typing. The program can use this text as input.

this - In many programming languages, the word **this** is used to refer to the main object that the code is in.

Title Screen - A **title screen** introduces a user to the program. Often, it's the first screen the user sees. A title screen says the name of the program and who created it. Sometimes it will say how to use the program, or when the program was made.

User - A **user** is someone who uses a computer program. When a computer programmer creates programs, they think about how the user will use their program.

using - The **using** command tells the computer which namespace you're using. Using a namespace lets you write C++ code in an easier way.

Value - A variable's **value** is the information that the variable is storing for the computer.

Variable - A **variable** is a place for a computer to store information.

Variable Declaration - A **variable declaration** is a way of telling the computer that you're using a variable, and what type of variable it is.

Variables can be different kinds of numbers, or different kinds of letters and numbers combined. Programmers declare the type of variable so the computer won't get confused.

A declaration for a float variable looks like this:

```
float Variable_Name;
```

Variable Initialization - Programmers **initialize a variable** by giving it a value when they declare it. When programmers don't know what value to initialize a variable with, they usually use zero (0).

Variable Naming - Variables can only be named in certain ways. You'll use valid variable names whenever you declare a new variable. Valid variable names:

- Only use characters a-z, A-Z, 1-9, or _
- Are not longer than 32 characters
- Do not start with 1-9
- Do not use special characters, such as: ! - $ [

- Do not use spaces
- Are not the same as a C++ command

void - The word **void** tells the computer that a function won't return a value.

while ( ) - A **while ( ) loop** is a type of loop that repeats while a condition is true. For example:

```
while ( condition )
{
    // Code in this section will be repeated as long
    // as the condition is true
}
```

Windows Forms Project - A **Windows forms project** is a type of C++ project that lets people use parts of the Windows interface to interact with the program. Windows forms projects use windows, buttons, menus, and other things the user can click on.

Windows Interface - The **Windows interface** is how you normally interact with your computer. It includes windows, buttons, menus, and other things you can click on.

Writing to Files - A computer **writes** information to a file by storing information in the file. See **Reading from Files**.

XmlDocument - An **XmlDocument** object is a C++ object designed to hold XML data. The code for creating an XmlDocument is similar to creating a DataSet, or any other data managed object.

```
Class ^ObjectName = gcnew XmlDocument ( );
```

XML Element - An **XML element** is a label for information in an XML file. XML elements have opening and closing **tags**, similar to how C++ has opening and closing curly brackets.

The opening tag looks like: `<element_name>`

The closing tag has a / before the element's name, like: `</element_name>`

The element's information goes between the tags.

XmlElement - An **XmlElement** object stores information for one XML element.

```
XmlElement ^Object_Name = XmlDocumentName -> CreateElement ("ElementName");
```

The code for creating an XmlElement object looks like the example above. In the example, **Object_Name** is what you name your XmlElement object.

XmlDocumentName is the name of your XmlDocument object. **ElementName** is the name of the element.

XML File - XML files are text files that use XML to label the information in the text file. XML files have the file extension .xml. One of the advantages of XML files is that they can be edited like any other text file. This makes it easy for programming languages like C++ to store information in them.

Xml Namespace - The **Xml namespace** is a grouping of commands related to working with XML files and the XML language. You used the standard (std) namespace in previous projects to make it easier to type standard commands. The XML namespace makes it easier to type the commands you'll need for working with your XML file.