

Introduction to Programming with C++

TEAM 2080



What are some benefits to programming with C++ in FRC?

- **C++ is a much more efficient language than both LabVIEW and Java**
- **C++ is a more modern programming language and is used in most of the major robotics/STEM related companies like SpaceX and Tesla**
- **C++ can be compiled and deployed much faster than LabView, reducing program development time**



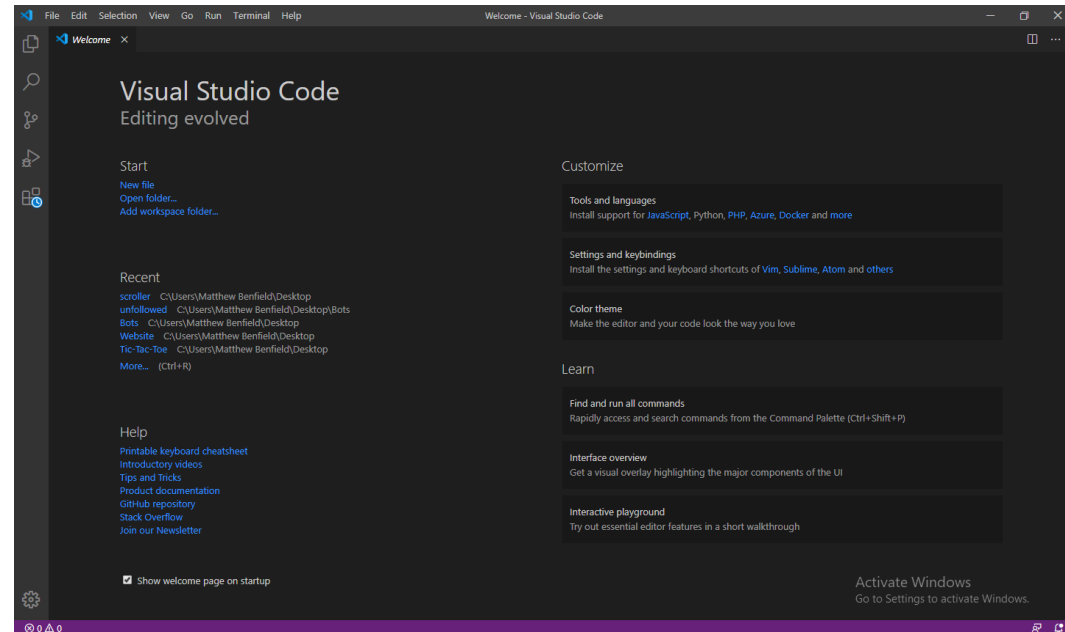
Setting Up Visual Studio Code

Installation:

Go to <https://code.visualstudio.com/> and download the correct version of VS Code for your device.

Go through the installation steps.

Once the installation is complete, open VS Code. You should be greeted with a page like this:



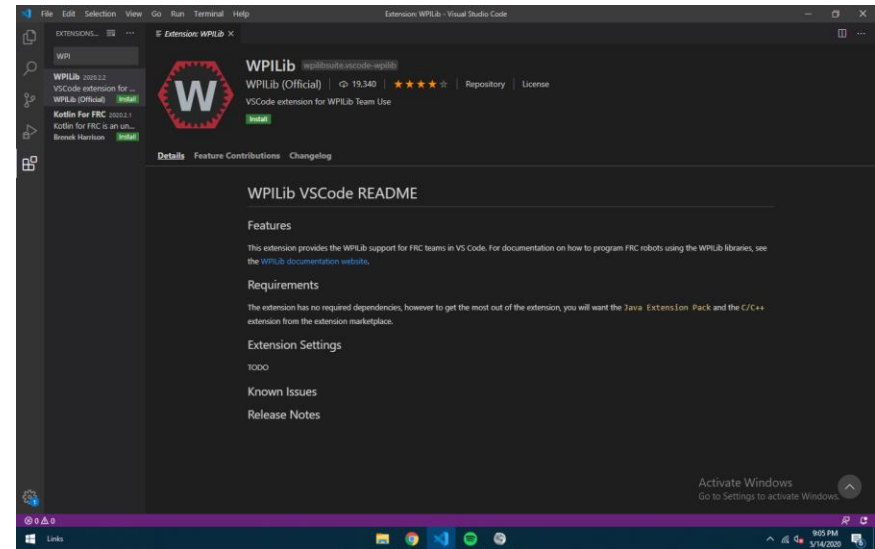
Setting Up Visual Studio Code

WPI Extension:

Click on the last button on the left nav bar. It looks like four squares with one drifting off.

Once on the Extensions page, search WPILib and click the first result. You should get brought to a page that look like this:

Click the green "Install Button". It will turn blue. VS Code is now ready for use with FRC.



Making Your First Program

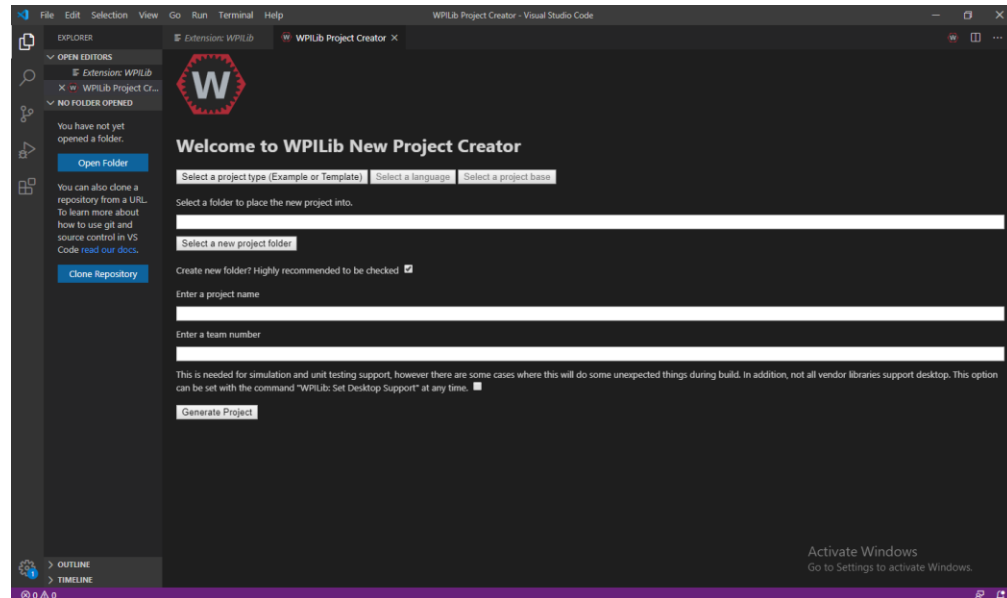
First Program Setup:

Click the little "W" in the top right corner. This will open the WPILib Command Palette.

Type "Create Project" and click the first link. You should be brought to a page that is similar to this:

Click "Select Project Type" then "Template" then "cpp" then "Timed Robot"

Fill out the rest of the form. Then click "Generate Project". WPILib will create a basic outline for your first program



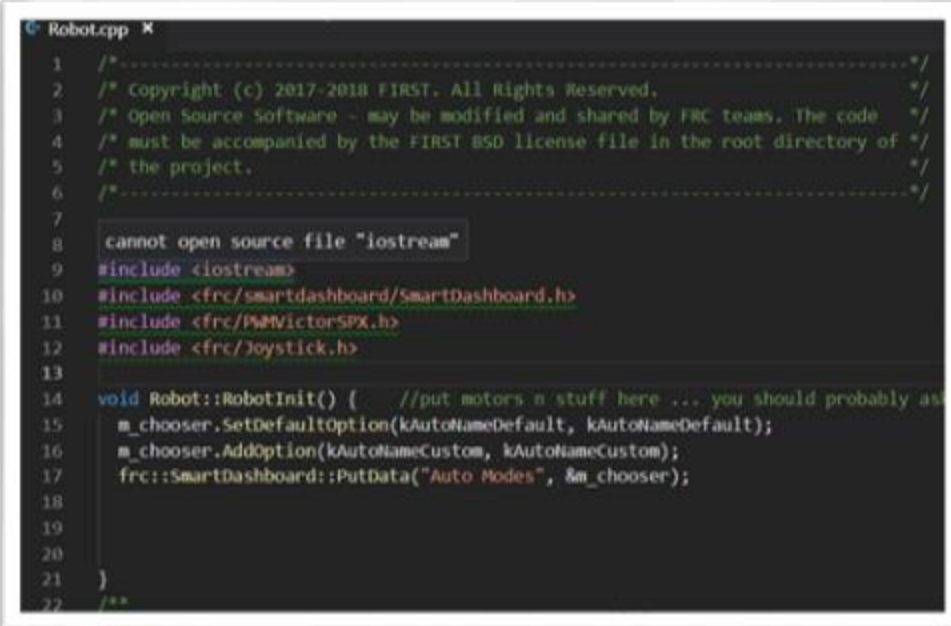
#includes

Includes are examples of libraries. In programming, a library is a collection of functions, files, programs, routines, or scripts that can be referenced to in the code. Think of it like writing an essay and going to the library to pull quotes from different authors.

Since vanilla C++ is so basic on its own, most FRC programs will begin with one or more includes. Some examples of these includes are:

```
#include <frc/Joystick.h>
#include <frc/TimedRobot.h>
#include <ctre/Phoenix.h>
#include "WPILib.h"
```

Computers will read a program from top to bottom. This is the reason why our libraries are declared at the beginning of our program, that way when the program comes across something from that library later in the program, it will know exactly what is it.



```
Robot.cpp
1  /*-----*/
2  /* Copyright (c) 2017-2018 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code */
4  /* must be accompanied by the FIRST BSD license file in the root directory of */
5  /* the project. */
6  /*-----*/
7
8  cannot open source file "iostream"
9  #include <iostream>
10 #include <frc/smartdashboard/SmartDashboard.h>
11 #include <frc/PWMVictorSPX.h>
12 #include <frc/Joystick.h>
13
14 void Robot::RobotInit() { //put motors n stuff here ... you should probably ask
15     m_chooser.SetDefaultOption(kAutoNameDefault, kAutoNameDefault);
16     m_chooser.AddOption(kAutoNameCustom, kAutoNameCustom);
17     frc::SmartDashboard::PutData("Auto Modes", &m_chooser);
18
19
20
21 }
22 /**
```

Data Types

When programming, all that is done is the manipulation of various data types through very precise instructions.

In C++ there are numerous different data types(variables) that can be manipulated. Some of these are listed below:

Name	C++ Representation	Range
character	char	-128→127
interger	int	+/-2 billion(whole numbers)
void	void	represents lack of data
string	string	any collection of characters
boolean	bool	true(1) or false(0)
floating point interger	float	+/- 3.4x10 ³⁸

Naming variables in C++:

Giving variables good names in an integral part of having "good" code. At one point or another someone else will look at your code, it is important that they can read and understand your code. For example,

Examples of declaring variables in C++:

```
int correction = 1.57;  
bool target = false;  
string robot = "Tim";
```

Controlling Program Flow

One advantage of programming is the ability to make unbiased decision based on the raw data along with the ability to do repetitive tasks.

In order to control the flow of the program we must be familiar with some different key terms. Each of the following term will be elaborated on:

- If
- If-Then-Else
- Switch
- While
- Do-While
- For

If Loop

When programming we may want a command to occur only if something happens. In programming we must break this down into different parts. There is a condition, if this condition is true, we want something to occur. Then there are the commands we want to occur.

The syntax for a if loop in C++ is:

```
if (condition) {  
    //run this command  
}
```

The condition can be any statement such as "a>b", "a==b", "a<=b", "a!=b". Conditions can also be combined to create more complex criteria:

```
if ( (a==b || (b==c) && (c==d) ) {  
    //run this command  
}
```

"||" means "or" and "&&" means "and"

Here are some example of working if loops:

```
if (auxiliary->GetRawButton(7)) {  
    ClimberTrack->Set(8);  
}
```

```
if (drive->GetRawButton(5)) {  
    Jacks.Set(frc::DoubleSolenoid::Value::lkForward);  
}
```

If-Else Loop

There may be times we want to do something to occur if the condition in the IF Loop is not true. This is where an If-Else Loop may come into play.

The syntax for an if-else loop in C++ is:

```
If (condition) {  
    //run this command if the condition is true  
}else{  
    //run this command if the condition is false  
}
```

An else may also be combined with an if to create an else if statement which allows you to check for multiple conditions:

```
If (condition) {  
    //run this command if the condition is true  
}else if(condition){  
    //run this command if the second condition is true  
}else if(condition){  
    //run this command if the third condition is true  
}
```

Switch

The Switch statement is a different type of the if-else statement. This statement was created to handle larger quantity of conditions.

The syntax for a Switch statement in C++ is:

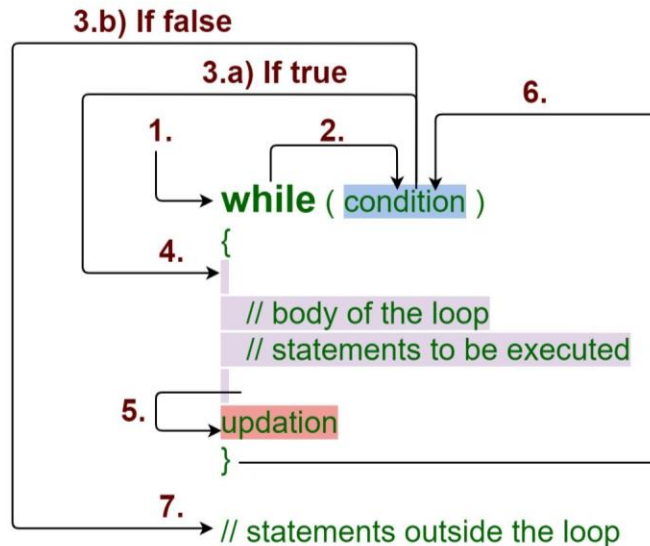
```
switch(variable) {  
    Case 0:  
        //run for case 0  
    Case 1:  
        //run for case 1  
    Case 2:  
        //run for case 2  
    Case 3:  
        //run for case 3  
    Case 4:  
        //run for case 4  
    Case 5:  
        //run for case 5  
    default :  
        //run if none of the above if true  
}
```

While

Whenever we want something to occur again and again until the condition is false, we would use a while loop.

The syntax for a While statement in C++ is:

```
while(a==b){  
    //will run as long as long as a is equal to b  
}
```



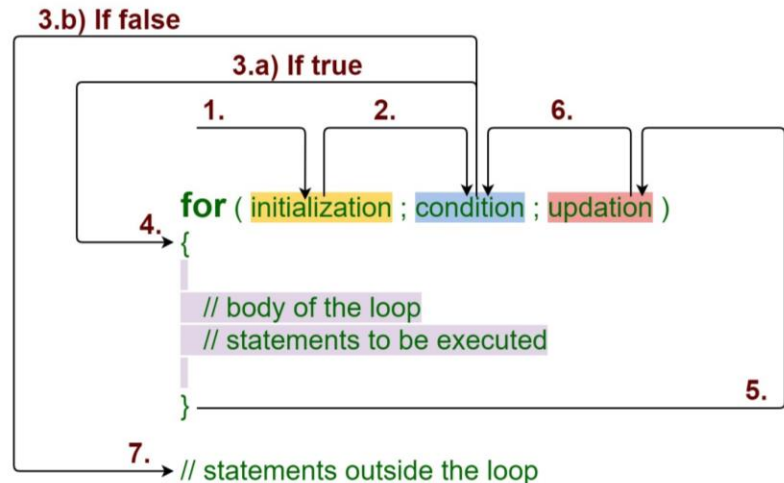
For

A common use for a while loop is to act as a counter:

```
int a;  
  
while(a<20 ){  
    a++;  
}
```

The designers of C++ knew that people would be using this code often, so they decided to simplify its process. They created a new type of loop called the For Loop:

```
for(int a, a< 20, a++){  
    //run this stuff  
}
```



Functions - Declarations

Functions are blocks of code than can be called to do certain tasks. There are two parts to a function, the declarations and the definition.

The syntax of a function is:

```
Void functionName () {};
```

What this means is that there is a function called functionName which returns no value(void). Anything that is put inside the curly braces is what will occur when the function is called.

Here are some examples of different functions:

```
void Jack() {  
    If(drive->GetRawButton(5)) {  
        Jacks.Set(frc::DoubleSolenoid::Value::kForward)  
    }  
}
```

```
void Elevator() {  
    Lift->Set(-auxiliary->GetY());  
}
```

```
void RobotDiffDrive() {  
    m_robotDrive.ArcadeDrive(drive->GetY(), drive->GetZ());  
}
```

Functions - Definitions

After defining a function, you must give the function purpose by defining what it must do.

The syntax of a function is:

```
Void functionName () {};
```

What this means is that there is a function called functionName which returns no value(void). Anything that is put inside the curly braces is what will occur when the function is called.

Here are some examples of different functions:

```
void Jack() {  
    If(drive->GetRawButton(5)) {  
        Jacks.Set(frc::DoubleSolenoid::Value::kForward)  
    }  
}
```

```
void Elevator() {  
    Lift->Set(-auxiliary->GetY());  
}
```

```
void RobotDiffDrive() {  
    m_robotDrive.ArcadeDrive(drive->GetY(), drive->GetZ());  
}
```

Joysticks

Joysticks are obviously very important in FRC. Obtaining the knowledge of how to use joysticks is integral to FRC programming.

To declare a gamepad write:

```
Joystick* exampleJoystick =  
new Joystick(0);
```

Note that the number in parentheses is the number that the controller is assigned to in the Driver Station. You can also drag each gamepad up or down in the driver station to change its place.

To use a joystick to control a motor, you simply replace the power number with the joystick value:

```
Lift->Set(-auxiliary->GetY());
```



When in the Driver Station, connect your gamepad(s) and click the usb button. Then click on the gamepad you want to read. When you click a button on the gamepad the corresponding button will turn green, simply count the number of the button then you can control it:

```
auxiliary->GetRawButton(1)
```


Declaring Motors

I think that anyone would agree that being able to control the motors on the robot is an integral part of programming. To be able to do this we first must declare a Speed Controller Object. This will tell the program what type of motor controller we are using, the name of the motor, and what port the motor is connected to on the roboRIO. The syntax for this is:

```
Jaguar*exampleJaguar = new Jaguar(0);
```

In this example we are using a Jaguar. Keep in mind that the syntax for declaring your specific motor controller may vary depending on what brand you buy. Check the manual and website of the vendor to make sure you use the correct syntax.

```
// Motor Controllers
// Victor Motor Controllers
WPI_VictorSPX * Lift = new WPI_VictorSPX(8); // Victor SPX Lift motor Victor #8
WPI_VictorSPX * fourBar = new WPI_VictorSPX(5); // Victor SPX Four Bar motor Victor #5
WPI_VictorSPX * intakeRoll = new WPI_VictorSPX(7); // Victor SPX Cargo Intake motor Victor #7
WPI_VictorSPX * climberTrack = new WPI_VictorSPX(10); // Victor SPX Climber Track Victor #10

// REV SPARK Max Motor Controllers
static const int leftLeadDeviceID = 1, leftFollowDeviceID = 3, rightLeadDeviceID = 4, rightFol
rev::CANSparkMax m_leftLeadMotor{leftLeadDeviceID, rev::CANSparkMax::MotorType::kBrushless}; // Sets
rev::CANSparkMax m_rightLeadMotor{rightLeadDeviceID, rev::CANSparkMax::MotorType::kBrushless}; // Se
rev::CANSparkMax m_leftFollowMotor{leftFollowDeviceID, rev::CANSparkMax::MotorType::kBrushless}; //
```

Motor Speed

Once declaring the motor controllers , we will want to send them power, or lack thereof. A speed controller object takes a single speed parameter varying from -1.0 (full reverse) to $+ 1.0$ (full forward).The syntax for sending power is :

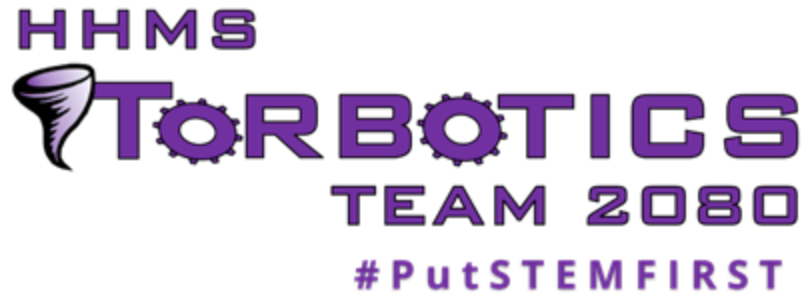
```
exampleJaguar->Set(0.7)
```

In this example we are using a Jaguar.

```
void FourBar()
{
    if(auxiliary->GetRawButton(1))
    {
        fourBar->Set(8);
    }
    else if(auxiliary->GetRawButton(2))
    {
        fourBar->Set(-1);
    }
    else
    {
        fourBar->Set(0);
    }
}
```

Credits

- This lesson was written by FRC 2080 for FRCTutorials.com
- You can contact the author at frcteam2080@gmail.com or www.torbotics2080.org
- Please visit our GitHub at <https://github.com/frcteam2080> to see our robot code.



- More lessons for FIRST Robotics Competition are available at www.FRCTutorials.com



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).