C Programming For FIRST FRC Competitions

Keith M. Hughes

DON'T PANIC!

easyC Pro come with lots of examples and tutorials. Look at them with copies of these slides by your side and you will see how it all fits together.

Programming... Who Needs It?

Every team will most likely need some programming. It is necessary to connect the controls on the Operator Interface to the motors, relays, etc, on the robot.

Autonomous mode requires the most programming, though it doesn't necessarily have to be that sophisticated.

Warning

Don't wait on understanding the programming or thinking about it until towards the end of the build season. The programmers need to be involved early on with the planning for hardware since they have to write programs for the sensors, because they have to think about what can and cannot be done in software.

This means the programmers need to understand programming sooner than later.

What We'll Cover

The basics of the C programming language.

I am not telling you everything as I am trying to keep this a short introduction. Experienced C programmers will notice that some things I am saying have to happen don't always have to, but it takes too long to explain everything.

Variables

Variables are named bits of memory in the processor. You use variables to keep bits of computations that you have done or to control the robot.

```
pwm1 = p1_x;
pwm2 = 255 - p1_y;

speedLeftMotor = GetAnalogValue(3) * 2;
speedRightMotor = speedLeftMotor / 2;
```

The variables above are `pwm1`, `p1_x`, `pwm2`, `p1_y`, `speedLeftMotor`, and `speedRightMotor`.

Variables

Variables in C must be given a type. The type says what kind of information can be stored in the variable.

The type for a variable is given in a variable declaration.

```
unsigned char speedLeftMotor;
int leftWheelCounts;
```

The underlined parts are the type of the variable. The rest is the name of the variable.

Note: Notice the semicolon at the end. Don't forget this.

Variable Types

`unsigned char`	Value from 0 to 255
`char`	Value from -128 to 127
`unsigned int`	Value from 0 to 65535
`int`	Value from -32768 to 32767
`long`	Value from -big to other big

There are more, but these are the ones you will usually use. `unsigned char` is for controlling motors and reading analog values. Some sort of `int` or `long` is good for reading wheel or gear encoders.

Variables

Variables must be declared before they are used. It is an error if they are used before being declared.

The declarations must happen at the top of a function or subroutine, or in the parameter list of a function or subroutine.

C Is Case Sensitive

C is case sensitive. Capital letters are considered different than small letters.

This means all of the following are the names of different variables.

```
motorSpeed
MotorSpeed
MoToRsPeEd
MOTORspeed
```

Try not to distinguish variables by the case of their letters. This would make it difficult to read your program and hard to find errors.

FIRST

Assignment

Assignment gives a variable a value.

```
variable = expression;
```

```
motorSpeed = 127;
pwm1 = 255 - p1_x;
value = 2 * Limit(2 * GetAnalogValue(3), 255);
shouldThrowBall = operatorSwitch1 & limitSwitch2
```

Expression Operators

Expressions consist of variables, numbers, and function calls, possibly put together with the following operators.

These are good for analog values (when not just 0 or 1, like joystick input):

*	Multiplication
/	Division
+	Addition
-	Subtraction

```
1 + 2 * 3
pwm1 + 4
```

Expression Operators

Expressions consist of variables, numbers, and function calls, possibly put together with the following operators.

These operators are good for digital values (only has value of 0 or 1):

&	And. 1 if both sides are 1. Otherwise is 0.
|	Or. 0 if both sides are 0. Otherwise is 1.
!	Not. Makes 0 into 1 and 1 into 0.

```
operatorInterfaceSwitch1 & operatorInterfaceSwitch2
limitSwitchWinch | operatorInterfaceSwitch2
!operatorInterfaceSwitch4
```

Condition Expressions

Also called boolean or comparison expressions. Have a value of true or false.

Equality operators are

`==` Both sides are equal

```
a == b + 1
GetDigitalInput(3) == CLOSED
```

`!=` Both sides are not equal to each other

```
a != b + 1
GetDigitalInput(3) != CLOSED
```

Beware, equality is checked by `==`, not `=` (`=` is for assignment). The following will either give an error from the compiler or not do what you want.

```
a = b + 1
GetDigitalInput(3) = CLOSED
```

FIRST

Condition Expressions

Other boolean operators are

- `<` Left side is less than the right side
  ```
  a < b + 1
  GetAnalogInput(3) < 100
  ```

- `<=` Left side is less than or equal to the right side
  ```
  a + 7 <= b + 1
  GetAnalogInput(3) <= 100
  ```

- `>` Left side is greater than the right side
  ```
  a > 2 * (b + 1)
  GetAnalogInput(3) / 2 > 100
  ```

- `>=` Left side is less than or equal to the right side
  ```
  a >= b + 1
  GetAnalogInput(3) >= 100 + GetAnalogInput(4)
  ```

FIRST

Condition Expressions

Can combine boolean expressions with boolean connectives.

`&&` And. True if both sides are true. Otherwise false.

```
(a < b + 1) && (GetAnalogInput(3) < 100)
```

`||` Or. False if both sides are false. Otherwise true.

```
GetDigitalInput(3) == CLOSED || GetAnalogInput(3) <= 100
```

`!` Not. Makes false into true and true into false.

```
!((a < (b + 1)) && (GetAnalogInput(3) <= 100))
```

Can get fairly complicated.

```
((a < b + 1) && (GetAnalogInput(3) < 100)) || (a < 4)
```

Conditionals

Conditionals evaluate a condition expression (remember, these have a value of true or false). The conditional will execute a block of code (called the body of the conditional) if the value is true.

```
if ( condition ) {

    ...statements to do if condition has a true value...
    ...(called the body of the conditional)...

}
```

Conditionals

If the button is pushed, stop the motors and throw the ball. Always run the motors from the Operator Interface.

```
if ( GetDigialValue(3) == CLOSED ) {
    SetMotors(127);
    ThrowBall();
}

Arcade4 (1, 1, 1, 2, 1, 2, 3, 4, 0, 0, 0, 0);
```

FIRST

Conditionals

Can have a series of

```
else if
```

statements. If condition 1 is true, its body will be executed. All conditionals below it will be skipped. If condition 1 is false, condition 2 will be evaluated. If condition 2 is true, its body will be executed, and all other conditionals below it will be skipped. Etc.

```
if ( condition1 ) {
    ...done if condition 1 is true...
}
else if ( condition2 ) {
    ...done if condition 2 is true...
}
else if ( condition3 ) {
    ...done if condition 3 is true...
}
```

Conditionals

If one switch is closed, stop the motors and throw the ball. Otherwise, if the other switch is closed, pick up a ball. Always run the motors from the Operator Interface.

```
if (GetDigitalInput(4) == CLOSED) {
    SetMotors(127);
    ThrowBall();
}
else if (GetDigitalInput(5) == CLOSED) {
    PickUpBall();
}

Arcade4 (1, 1, 1, 2, 1, 2, 3, 4, 0, 0, 0, 0);
```

Conditionals

Can end a conditional series with an

```
else
```

statement. The body of the else will be done if all conditions above it are false.

```
if ( condition1 ) {
    ...done if condition 1 is true...
}
else if ( condition2 ) {
    ...done if condition 2 is true...
}
else if ( condition3 ) {
    ...done if condition 3 is true...
}
else {
    ...done if none of the conditionals above are true...
}
```

FIRST

Conditionals

If one switch is closed, stop the motors and throw the ball. Otherwise, if the other switch is closed, don't change the motors and pick up a ball.

Now the motors will be run from the Operator Interface only if neither switch was closed.

```
if (GetDigitalInput(4) == CLOSED) {
    SetMotors(127);
    ThrowBall();
}
else if (GetDigitalInput(5) == CLOSED) {
    PickUpBall();
}
else {
    Arcade4 (1, 1, 1, 2, 1, 2, 3, 4, 0, 0, 0, 0);
}
```

FIRST

Conditionals

Can have assignment statements, calls to functions (subroutines), other conditionals, and loops inside of the body of a conditional.

Remember to put the { } around the body of the conditional.

Loops

Loops provide a way to repeat a group of statements over and over again until some condition is met.

```
while ( condition ) {

    ...statements to repeat (called the body of the loop)...

}
```

where `condition` is a condition expression (remember, these have a true or false value).

The body of the loop will be repeated while the condition expression is true.

Put curly braces around the body.

Loops

Gradually increase the speed of a motor while a switch is open. When the switch closes, the motor will turn off.

```
speed = 130;
while (GetDigitalValue(3) == OPEN) {
    speed = speed + 1;
    pwm1 = speed;
}
pwm1 = 127;
```

Loops

Start some event and use the loop to wait until some other event happens, then do something else.

```
SetMotor(WINCH_MOTOR, 178);

while(GetAnalogInput(WINCH_POT) < 200) {
}

SetMotor(WINCH_MOTOR, 127);
```

Loops

Loops can count to repeat a group of statements some number of times.

```
count = 1;
while (count <= numberBalls) {
    throwBall();

    count = count + 1;
}
```

Bad if write the following. Why?

```
count = 1;
while (count <= numberBalls) {
    throwBall();
}
```

Loops

The previous loop

```
count = 1;
while (count <= numberBalls) {
    throwBall();

    count = count + 1;
}
```

can be written as

```
for (count = 1; count <= numberBalls; count = count + 1) {
    throwBall();
}
```

Loops

Sometimes you want loops that run forever.

```
while (1 == 1) {
    if (GetDigitalInput(4) == CLOSED) {
        ThrowBall();
    }
    else if (GetDigitalInput(5) == CLOSED) {
        PickUpBall();
    }

    Arcade4 (1, 1, 1, 2, 1, 2, 3, 4, 0, 0, 0, 0);
}
```

FIRST

Loops

Can have assignment statements, calls to functions (subroutines), conditionals, and other loops inside of a loop.

Remember to put the { } around the body of the loop.

Functions

Named sections of code that can be called from other sections of code.

Also called subroutines.

Every executable statement in C must live in a function.

Functions

Function call

```
pwm1 = limit((px_1 - 127) * 2 + 127);
```

Function definition

```
int limit(int value) {
    if (x > 255) {
        return 255;
    }
    else if (x < 0) {
        return 0;
    }
    else {
        return value;
    }
}
```

The statements inside the function are called the body of the function.

Functions

```
count = 1;
while (count <= numberBalls) {
    SetMotor(WINCH_MOTOR, 180);
    while (GetAnalogValue(WINCH_POT) < 200) { }
    SetMotor(WINCH_MOTOR, NEUTRAL);

    SetMotor(THROW_MOTOR, 255);
    while (GetDigitalValue(THROW_LIMIT_SWITCH_THROW) == OPEN) { }
    SetMotor(THROW_MOTOR, NEUTRAL);

    SetMotor(THROW_MOTOR, 0);
    while (GetDigitalValue(THROW_LIMIT_SWITCH_ARMED) == OPEN) { }
    SetMotor(THROW_MOTOR, NEUTRAL);

    SetMotor(WINCH_MOTOR, 0);
    while (GetAnalogValue(WINCH_POT) > 0) { }
    SetMotor(WINCH_MOTOR, NEUTRAL);

    count = count + 1;
}
```

Functions

```
count = 1;
while (count <= numberBalls) {
    raiseWinch();

    throwBall();

    resetBallThrower();

    lowerWinch();

    count = count + 1;
}
```

Much easier to read and understand.

Functions

```
void raiseWinch(void) {
    SetMotor(WINCH_MOTOR, 180);
    while (GetAnalogValue(WINCH_POT) < 200) { }
    SetMotor(WINCH_MOTOR, NEUTRAL);
}

void throwBall(void) {
    SetMotor(THROW_MOTOR, 255);
    while (GetDigitalValue(THROW_LIMIT_SWITCH_THROW) == OPEN) { }
    SetMotor(THROW_MOTOR, NEUTRAL);
}

void resetBallThrower(void) {
    SetMotor(THROW_MOTOR, 0);
    while (GetDigitalValue(THROW_LIMIT_SWITCH_ARMED) == OPEN) { }
    SetMotor(THROW_MOTOR, NEUTRAL);
}
```

FIRST

Function Prototypess

Description of the function.

Usually placed in .h files.

A prototype is needed if the function is used before it is defined, or if it is written in one .c file and used in a different .c file.

```
void raiseWinch(void);

void throwBall(void);

void resetBallThrower(void);

int limit(int value);
```

FIRST

Some Useful Terms

Compiler

Turns C program into the machine language for the controller. Has a preprocessing phase (for #define).

```
pwm1 = limit(p1_x*2, 255);
```

Machine Language

What the robot controller actually understands. Found in .HEX files.

```
10110100
11100101
00001011
```

Loader

Loads the machine language output of the compiler (along with other stuff) into the robot controller.

FIRST

Macros

Allows programmer to create aliases for variables, constants, or expressions which make a program easier to read.

The compiler replaces the aliases with their values before the compiler produces machine code.

Definitions usually placed in .h files.

Not necessary, but usually the alias names are written in all capital letters.

Macros

Can give built-in variables a name that is easier to understand.

The definition

```
#define FRONT_LEFT_MOTOR   pwm1
#define FRONT_RIGHT_MOTOR  pwm2
```

In your program code

```
FRONT_LEFT_MOTOR = 127;
FRONT_RIGHT_MOTOR = 173;
```

The compiler sees

```
pwm1 = 127;
pwm2 = 173;
```

FIRST

Macros

The programmer can give numbers a meaning.

The definition

```
#define NEUTRAL  127
```

In your program code

```
SetMotor(2, NEUTRAL);
```

The compiler sees

```
SetMotor(2, 127);
```

Macros

Macros can have one or more arguments.

The definition:

```
#define SPEED(speed)    (NEUTRAL + (speed))
```

In your program code:

```
setMotorSpeeds(SPEED(50));
```

What the compiler sees (if NEUTRAL defined as before):

```
setMotorSpeeds((127 + (50)));
```

FIRST

Comments

Human readable descriptions of what the program is doing.

The computer does not pay attention to these comments, they are only for humans reading the code. The compiler throws them away.

```
speed = SPEED(30);  // This comment goes to end of line.

/*
    This comment covers several lines until the bottom
    asterix slash. The comments to the right of the 3
    and 4 are legal, but makes the code hard to read.
*/
SetMotor(3 /* Left motor */, speed);
SetMotor(4 /* Right motor */, speed);
```

Things To Remember

Use semicolons where necessary. Use after variable declarations, assignment statements, function (subroutine) calls, function prototypes.

Do not use semicolons after #defines or comments or function definitions.

Conditionals and loops should have curly braces

```
    {
```

and

```
    }
```

around the body of the conditional or loop.

General Programming Tips

Think about the problem you are trying to solve before you start typing. Put yourself in the robot's shoes (or wheels, or treads). If you only had the sensors that the robot has, how would you accomplish the task?

General Programming Tips

Make your program readable, like a book.

Make use of macros and functions (subroutines).

Indent your program consistently if you are using a freeform editor.

Name variables and functions so that they make sense.

```
unsigned char leftMotorSpeed;
void raiseArmToThrowPosition(void);
```

You want to minimize the thinking necessary to read the code. Your great-grandfather, who grew up with Model T Fords, is going to read your program and you don't have time to explain it.

General Programming Tips

Comment your code. However, don't say exactly what the code is doing if it is simple, people can read that for themselves.

Not so great.

```
// Add one to age.
age = age + 1;
```

Much better.

```
// The robot is a year older.
age = age + 1;
```

No need to comment every single line.

More Information

http://www.ifirobotics.com

The company that makes the Robot Controller and the Operator Interface.

http://www.chiefdelphi.com

A forum for FIRST teams to discuss all aspects of FIRST.

Use your favorite search engine to look up *C Programming*, or *C Tutorial*.

http://robotics.hideho.org/

Look for the Programming In RobotWorld online book. Uses a BASIC-like language, but the programming concepts are the same.

FIRST

Have Fun!