**Vex Robotics Beginner Guide Programming**

*This is an extension of the regular guide and is for new people to learn basics terms and concepts (specifically VEX EDR) of programming in robotics and does not substitute a mentor so feel free to ask questions to learn more advanced information and processes.*

## Vex Robotics directly supports the programming IDE, RobotC. Alternatives do exist (PROS, easyC, Robot Mesh Studio, and others), but they are rarely used. The V5 might change up the usage of IDEs, plus a lot of teams are moving to PROS.

### Boolean Logic

Boolean logic is a type of systematic mathematics where the output can be expressed in 2 formats; True or False.

There are 3 main operators in boolean logic; “And” gates, “Or” gates, and “Not” gates

There are other gates such as NAND and XOR, but they are harder to program into systems and are not useful in the scope of vex robotics and therefore will not be talked about

|  |  |  |
| --- | --- | --- |
| Gate (RobotC syntax) | What it does | example |
| And (&&) | Exports true if ALL conditions in the system are true. If one conditions is false, it exports false |  |
| Or (||) | Exports true of AT LEAST ONE of the conditions is true  |  |
| Not (~) | Exports the opposite of the input |  |

### Variable types

A variable is a self-defined term that stores an value. This “value” can be one of 3 types; a string, a float, and an integer

IN C, there is not a “clean” way to turn one variable into another. There are some “hacks” to do it, but nothing built in to do it

|  |  |  |
| --- | --- | --- |
| Type of variable (robotC syntax) | What it does | Syntax in code |
| Integer (int) | Stores an integer [any number without a decimal] | int x =[an integer]; |
| Float (float) | Stores any real number. It can have a decimal | float x = [a number] |
| String (string) | Stores a set of characters. Strings are virtually never used in the robotC platform because of the lack of graphics. | String x = “[insert text here]” |
| Byte  | Creates an array that is separated by commas and remains functional when written on multiple lines | byte x [number of elements]= (1,2,3,4…..) |

### Built in commands into RobotC

To do any real programming in robotC, you need to know these lines, memorize them

There are more but these are the major ones

|  |  |
| --- | --- |
| **Command** | **What it does** |
| motor(port number or name) = (speed); | It sets the motor in slot one of the cortex to speed, range is [-127,127] |
| wait1msec(time); | Run the action for the input in milliseconds and then turn the off |
| SensorValue(sensor); | References the value of (sensor) |
| vexRT[] | References the remote control |
| ch(number between 1-4) | Use with vexRT[] and refers to a joystick axis |
| btn[(between 5 and 8)(U,R,D,L)] | Use with vexRT[] and refers to button coding |

### Loops

* Loops are used to make a section of code repeat until a constraint is met
* Usercontrol needs a while(true) around the entire code while Autonomous does not
* There are 3 main kinds of loops

|  |  |  |
| --- | --- | --- |
| Type of loop | What it does | Syntax |
| if/else loop | * It is good for statements that are multi conditional. One must have more than 2 states.
 | if(am\_\_16){Person can drive();}else if(am\_18) {Person can drive+vote();}else{Person can drive+vote+drink();} |
| While loop | This loop goes an action only until the action is no longer true. | while(sensorvalue[s1]>100) { motor(port1)=127; } |
| For loop | * Repeats a section of code a specific amount of time
* Read it something like this:

“While x is less then 100, increasing from 0 by 1 every time the code is run” | for(int x=0;x>100;x++) { motor(port1)=127; wait1msec(100); } |

* How to download code onto the cortex
* Can connect straight from the computer to the cortex or these following methods



### General Tips to Programming

1. If something isn’t working, be rational. If that doesn't fix it, google the problem
2. Sensors are always more accurate than wait statements
3. Always zero out your encoders before you start
4. Clear all motors after the while loop (or at least all the modified ones)
5. The more precision you can get. The better

**Pid:**This is a relatively advanced concept that deserves a guide in and of itself to explain so I will paste pseudocode here in the meantime. More information will be provided in [**https://tinyurl.com/GSMSTPID**](https://tinyurl.com/GSMSTPID)but for now, you can replace all the blue parts appropriately to make it work

int PID(int goal, int goal\_error)

{

 int power;

 int n;

 while(n == 1)

 {

 float kp=1 /\*number between 1 and 10\*/;

 float ki=1 /\*really small number\*/;

 float kd=1 /\*number\*/;

 int integral;

 int derivative;

 int PrevError;

 int error = goal-SensorValue(dgtl1/\*The proper sensor for the scenario\*/);

 integral = integral + error;

 if (integral >127)

 {

 integral=127;

 }

 derivative = error-PrevError;

 PrevError=error;

 power=(kp\*error)+(ki\*integral)+(kd\*derivative);

 if(abs(error)<goal\_error)

 {

 n=0;

 }

 }

 return power;