Just a few things to keep in mind. When writing a statement in a programming language you often times see things like:

while() {

// code

// code

}

Where there is an opening curly bracket: {

And a closing curly bracket: }

Paired together like: function() { }

Think of these brackets like happy friends high fiving each other! Just remember friends never leave each other hanging so if you open a curly bracket always close it!

Also sometimes you’ll have to put one curly bracket pair inside another like

if() {

while() {

// code

// code

}

}

Just remember that each open curly bracket needs to be closed

Also keep in mind whenever you write a line of code that issues a command to the program you have to end the line with a semicolon like:

printToScreen(“Hello!”);

Or:

ProgramStop();

Each line that you write that’s issuing a command to the program must end with a semicolon.

One last note about how to type your program. Certain commands like ControllerDriveForward() or printToScreen() want you to input something in the parenthesis, the (), like you did with the printToScreen(“Hello, World!”); command we discussed. In the robot-config.h file you will see listed the various commands I’ve made for you to use like:

void ControllerTurnLeft(int);

What that is showing you is that you have a command available for you to put in your program and you issue that command in your program by ignoring the ‘void’ and just typing the actual command. You also need to replace anytime you see ‘int’ with an actual number. So for instance if you find a command listed as:

void ControllerDriveBackward(int);

You would type that in your program inside int main() like:

Int main() {

while() {

// code

ControllerDriveBackward(100);

// more code

}

}

Notice how the ‘int’ in the original command is replaced when you typed it with a number. Numbers are either going to represent a motor speed or an amount of time. I’ll explain more about specific commands in specific instruction steps below.

One more thing to note all the controller programming you type needs to be inside your programs int main() { while() { loop. So all your code you should be typing like this:

Int main() {

while(OnOff) {

// code

// code

// more code

ProgramStop();

}

printToScreen(“Program Stopped”);

}

Except for the line where you type after the while loop ‘printToScreen(“Program Stopped”); This is so you can have the vex brain screen let you know you’ve successfully exited the while loop.

Controller Programmers:

The goal is to get all the major functions of the robot assigned to specific buttons on the controller. So things like driving forward will be assigned to D-Pad button up or turning left will be assigned to D-Pad button left. There are individual buttons for the claw arm up and down as well as individual buttons for closing and opening the claw. There is also a button that stops the program, right now that button is ‘X’ and that is what the ProgramStop() function listens for. When ProgramStop() detects ‘X’ being pressed, the program exits the while(OnOff){} loop and stops listening for controller input.

Programmers are responsible for certain aspects of the actual build of the robot. The programmers need to tell the program which port has been assigned to which motor, if port 3 or port 4 has been assigned to the arm motor in your program for example. So it’s on the programmer to tell the builders which port they need to put which motor based on how they programmed them. If you use the default assignments I made in the robot-config.h file ports 1 and 2 are the wheel motors, port 3 is the arm motor and port 4 is the claw motor. Also it’s the responsibility of the programmers to let the other team members know how to use the controller, which buttons do what actions.

A few other things to note before we get into the instructions:

The commands I’m asking you to use for claw arm up/down and claw open/close have preset settings I’ve defined that limit the amount of rotation and the speed of rotation the motors have to prevent you from damaging the arm or claw by trying to make the arm go up higher than it is able or the claw open or close farther than it’s able. I GUESSED when I defined those settings. It’s very likely you’ll have to make adjustments in your programming to those settings once the robot is built to make the arm and claw motors match what your individual robot is able to do. I’ll go over this in more detail when I outline those steps in the instructions.

One last thing regarding the drive wheel motors - I also GUESSED which orientation the motors on your individual robot will have. Meaning you might find that turn left actually turns right on your final build or drive forward button might go backwards. Again that just means a very simple adjustment you’ll need to make in your program to match your individual programming to your individual robot. It’s no big deal and I’ll also out line specifically how to adjust that in the appropriate instruction step.

\*\*\* INSTRUCTIONS \*\*\*

Week one controller programming goal:

Have person A’s program include the following controller functions

* Assign a controller button to each of these individual commands
  + Drive forward
  + Drive backward
  + Turn left
  + Turn right
  + Move arm up
  + Move arm down
  + Close claw
  + Open claw
  + Button to stop program
* There are optional challenge steps you can choose to have a go after you’ve proven a working set of basic commands. These are totally optional and up to your team if you want to try them. The commands I’m giving you will allow your robot basic functionality but completing challenge steps will let you expand on those basic actions in various ways.
* Preparing the program

All code needs to be typed inside Person A’s int main() { //code } like:

int main() {

// code

// code

}

The main() *function* is used to store all the commands and your commands are packaged between the opening curly bracket { and closed curly bracket }

When you press ‘play’ to start your program on your robot brain it starts at the top line inside the curly brackets of int and goes line by line down until it reaches the closing curly bracket ending your program

Next we need a part of the program that will listen for controller input for an indefinite period of time. If we didn’t do this the program would listen exactly one time for the buttons being pressed and rush straight to the end of the program in about 30 milliseconds, which is about .03 seconds!

To keep the program listening for input from the controller your program needs a while() { } loop. The parenthesis in the while loop is checked whether or not the statement inside is ‘true’ or ‘false’ and if true the code inside the loop keeps executing until the parenthesis are checked as false. When the while() is checked as false the program exits the while() loop’s { } curly brackets and continues executing code line by line after the while() loop’s closing curly bracket.

So to set up the program’s while() loop type:

int main() {

while(OnOff) {

}

}

The ‘OnOff’ is a variable that is by default ‘true’ meaning it will allow the program to loop inside the while loop. But if ‘OnOff’ could only be ‘true’ the program could never exit the loop creating what’s called an infinite loop which is bad news.

So to prevent the while() loop from being an infinite while() loop you need to add a command to the line right before the closing curly bracket } of the while loop:

ProgramStop();

This tells the program to listen on the controller for button ‘X’ so that if button ‘X’ on the controller is ever pressed ProgramStop() will automatically change the ‘OnOff’ inside the while() loop from ‘true’ to ‘false’ allowing the program to exit the while() loop.

One more thing to note about any program loop, the computer will try to rush through the loop as fast as it possibly can which can create problems for other programs and other parts of the computer that want to use the processor for their own commands. To solve this we need to tell the loop to wait for just a bit after each time the program goes through the loop. We do this by using the pauseTask() command which is listed as:

void pauseTask(int);

Remember don’t type ‘void’ and replace ‘int’ with your own number when typing the command in your own program. The value ‘int’ takes milliseconds where 1000 milliseconds is equal to 1 second so 100 milliseconds or 0.1 seconds is plenty of time to give our program a chance to let other tasks perform some commands.

So type in the line just before the ProgramStop(); command:

pauseTask(100);

Now we just need to make sure if we’ve pressed ‘X’ that the ProgramStop() command does actually exit the while() loop. We do this by telling the robot brain screen to print to the screen that the program has stopped. And to confirm that the command is being executed after the program has exited the while() loop we need to type this command just after the line that contains the while() loop’s closing curly bracket }. So type the printToScreen(“After while loop”) command line:

int main() {

while(OnOff) {

//code

//code

}

printToScreen(“After while loop”);

}

OK step 1, program prep is finished. The rest of the commands are going to be typed in the while() loop’s curly brackets { // code } so the program will listen for the button presses we’re about to assign.

This step may seem fairly quick and straightforward but remember you’ll have to check to make sure all these steps actually work and you’ll have to make some adjustments to these commands to fit your individual robot. And if you still have time after all that you can complete some challenge suggestions I’ll outline as well.

To recap the program will need these robot functions assigned to individual buttons:

* Drive forward
* Drive backward
* Turn left
* Turn right
* Claw arm up
* Claw arm down
* Claw close
* Claw open

To assign a command to button simply type in these commands I’ve written. But remember you will need to adjust the claw arm up/down and claw open/close commands to fit your individual robot.

Also remember not to type the ‘void’ and replace any ‘int’ in parentheses with your own numbers whose default range is between 15-100. This is the speed the motor will rotate in degrees per second.

void ControllerDriveForward(int);

* Default button is bound to D-Pad up

void ControllerDriveBackward(int);

* Default button is bound to D-Pad down

void ControllerTurnLeft(int);

* Default button is bound to D-Pad left

void ControllerTurnRight(int);

* Default button is bound to D-Pad right

void ControllerMoveArmUp();

* Default button is bound to joystick (3) up

void ControllerMoveArmDown();

* Default button is bound to joystick (3) down

void ControllerClawOpen();

* Default button is bound to joystick (4) up

void ControllerClawClose();

* Default button is bound to joystick (4) down

The parts of those commands you’ll likely need to adjust to fit your robot is found in the functions definitions section in the robot-config.h file. You’ll make the adjustments to the arm up/down and claw open/close commands in these definitions in the robot-config.h file:

void ControllerMoveArmUp() {  
 if(getAxisPosition(3) > 0) {  
 ArmMotor.startRotateTo(30, rotationUnits::deg, 15, velocityUnits::dps);  
 } else if(getAxisPosition(3) < 0) {  
 } else {  
 ArmMotor.stop();  
 }  
}  
  
void ControllerMoveArmDown() {  
 if(getAxisPosition(3) > 0) {  
 } else if(getAxisPosition(3) < 0) {  
 ArmMotor.startRotateTo(-30, rotationUnits::deg, 15, velocityUnits::dps);  
 } else {  
 ArmMotor.stop();  
 }  
}  
  
void ControllerClawOpen() {  
 if(getAxisPosition(4) > 0) {  
 ClawMotor.startRotateTo(30, rotationUnits::deg, 15, velocityUnits::dps);  
 } else if(getAxisPosition(4) < 0) {  
 } else {  
 ClawMotor.stop();  
 }  
}  
  
void ControllerClawClose() {  
 if(getAxisPosition(4) > 0) {  
 } else if(getAxisPosition(4) < 0) {  
 ClawMotor.startRotateTo(-30, rotationUnits::deg, 15, velocityUnits::dps);  
 } else {  
 ClawMotor.stop();  
 }  
}

The part that you would actually change is in the command:

ArmMotor.startRotateTo(30, rotationUnits::deg, 15, velocityUnits::dps);

* For the arm up/down commands

Or:

ClawMotor.startRotateTo(-30, rotationUnits::deg, 15, velocityUnits::dps);

* For the claw open/close

The ‘30’ or ‘-30’ in the command right after the opening parenthesis ( and before ‘rotationUnits::deg’ is the number you’ll likely have to change. This number, the 30 or -30 represents the degrees of rotation permitted for either the arm or claw. The 30 represents being able to open 30 degrees wide and the -30 then represents being able to close those 30 degrees by spinning the motor in the opposite direction. Depending on how your robot is built you might need to increase that number from 30 to say, 45 and similarly -45 if your robot arm/claw can move wider than the initial 30. Don’t over do this change you can damage the arm and claw if you do! Make small changes and check after each adjustment to be safe.

You’ll know your controller works when the buttons correctly control each aspect of the robot as expected. You’ll need to let the builders know how they need to wire the motors so it matches how you’ve programmed it.