VEX EDR Transition Guides
Moving Forward and Backward in ROBOTC Graphical and Modkit Block

In the code below, the robot is programmed to move forward for two rotations, stop and wait for two seconds, and then move backward for two rotations. The top code is written in ROBOTC Graphical for a VEX EDR Cortex robot equipped with two driving motors with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in Modkit Block for a VEX EDR V5 robot equipped with two Smart Motors.

Call Outs:
Program start
- In ROBOTC Graphical, the robot begins executing code at line 1.
- In Modkit, the robot begins executing code at the top of the `when (STARTED)` event.

- Move commands
In ROBOTC Graphical, the `forward()` command allows you to move the robot forward for a specified number of degrees, rotations, seconds, minutes, or milliseconds at a specified speed. The driving motors spin at the specified speed, slowing to a stop, and then braking at the end of the movement.

In Modkit, the `startRotateFor()` command allows you to turn on the previously specified motor for a specified number of degrees or revolutions. To move the robot forward, you can turn on each drive motor at the same speed and for the same positive duration.

- **Duration quantity**
  - In ROBOTC Graphical, the duration quantity is a number that allows you to specify how many degrees, rotations, seconds, minutes, or milliseconds to move the robot forward.
  - In Modkit, the duration quantity is a number that allows you to specify how many degrees or revolutions to move. Positive numbers spin the motor forward and negative numbers spin the motor backward.

- **Duration type**
  - In ROBOTC Graphical, the duration type allows you to specify whether the robot should move forward for degrees, rotations, seconds, minutes, or milliseconds.
  - In Modkit, the duration type allows you to specify whether the motor should move for degrees or revolutions.

- **Speed**
  - In ROBOTC Graphical, with “Simple Behavior” commands such as `forward()`, the speed is specified as a percent ranging from 0 to 100 (full speed).
  - In Modkit, the `setVelocity()` command allows you to set the speed of the previously specified motor as a percent. Any movements by the specified motor throughout the program will run at the specified speed.

- **Movement completion**
  - In ROBOTC Graphical, the program execution stays at the `forward()` command until its movement is complete, and then moves on in the program.
  - In Modkit, the program execution passes through the `startRotateFor()` command before the movement is complete. The `waitUntil()` command is set to hold the program execution until the drive motors have finished spinning, allowing the robot to complete the movement.

- **Wait**
  - In ROBOTC Graphical, the `wait()` command holds the program execution at that spot in the program for the specified duration.
  - In Modkit, the `wait()` command also holds the program execution at that spot in the program for the specified duration.
— In ROBOTC Graphical, the `backward()` command allows you to move the robot backward for a specified number of degrees, rotations, seconds, minutes, or milliseconds at a specified speed. The driving motors spin at the specified speed, slowing to a stop and then braking at the end of the movement.

— In Modkit, the `startRotateFor()` command allows you to turn on the previously specified motor for a specified number of degrees or revolutions. To move the robot backward, you can turn on each drive motor at the same speed and for the same negative duration.

- Duration quantity

— In ROBOTC Graphical, the duration quantity is a number that allows you to specify how many degrees, rotations, seconds, minutes, or milliseconds to move the robot forward.

— In Modkit, the duration quantity is a number that allows you to specify how many degrees or revolutions to move. Positive numbers spin the motor forward and negative numbers spin the motor backward.

**Key Differences**

In the VEX EDR Cortex system, the motors do not contain built-in encoders, so the Integrated Motor Encoders or Shaft Encoders must also be used with the code above. In the VEX EDR V5 system, all Smart Motors contain built-in encoders.

**VCS Commands Used:**

- `setVelocity()`
- `startRotateFor()`
- `waitUntil()`
- `wait()`
- `isDone()`
In the code below, the robot is programmed to turn right for two rotations, stop and wait for two seconds, and then turn left for two rotations. The top code is written in ROBOTC Graphical for a VEX EDR Cortex robot equipped with two driving motors with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in Modkit Block for a VEX EDR V5 robot equipped with two Smart Motors.

### Call Outs:

**Program start**

- In ROBOTC Graphical, the robot begins executing code at line 1.
- In Modkit, the robot begins executing code at the top of the `when(STARTED)` event.

**Move commands**

- In ROBOTC Graphical, the `turnRight()` command allows you to point turn the robot for a specified number of degrees, rotations, seconds, minutes, or milliseconds.
at a specified speed. The driving motors spin at the specified speed, slowing to a stop and then braking at the end of the movement.

— In Modkit, the `startRotateFor()` command allows you to turn on the previously specified motor for a specified number of degrees or revolutions. To have the robot point turn to the right, you can give the left motor a positive duration and the right motor a negative duration.

- **Duration quantity**
  - In ROBOTC Graphical, the duration quantity is a number that allows you to specify how many degrees, rotations, seconds, minutes, or milliseconds to spin the driving motors on the robot.
  - In Modkit, the duration quantity is a number that allows you to specify how many degrees or revolutions to spin the motor. Positive numbers spin the motor forward and negative numbers spin the motor backward.

- **Duration type**
  - In ROBOTC Graphical, the duration type allows you to specify whether the drive motors should spin for degrees, rotations, seconds, minutes, or milliseconds.
  - In Modkit, the duration type allows you to specify whether the motor should move for degrees or revolutions.

- **Speed**
  - In ROBOTC Graphical, with “Simple Behavior” commands such as `turnRight()`, the speed is specified as a percent ranging from 0 to 100 (full speed).
  - In Modkit, the `setVelocity()` command allows you to set the speed of the previously specified motor as a percent. Any movements by the specified motor throughout the program will run at the specified speed.

- **Movement completion**
  - In ROBOTC Graphical, the program execution stays at the `turnRight()` command until its movement is complete, and then moves on in the program.
  - In Modkit, the program execution passes through the `startRotateFor()` command before the movement is complete. The `waitUntil()` command is set to hold the program execution until the drive motors have finished spinning, allowing the robot to complete the movement.

- **Wait**
  - In ROBOTC Graphical, the `wait()` command holds the program execution at that spot in the program for the specified duration.
  - In Modkit, the `wait()` command also holds the program execution at that spot in the program for the specified duration.

- **Move commands**
  - In ROBOTC Graphical, the `turnLeft()` command allows you to move the robot point turn for a specified number of degrees, rotations, seconds, minutes, or
milliseconds at a specified speed. The driving motors spin at the specified speed, slowing to a stop and then braking at the end of the movement.

— In Modkit, the `startRotateFor()` command allows you to turn on the previously specified motor for a specified number of degrees or revolutions. To move the robot backward, you can turn on each drive motor at the same speed and for the same negative duration.

- Duration quantity

— In ROBOTC Graphical, the duration quantity is a number that allows you to point turn the robot for a specified number of degrees, rotations, seconds, minutes, or milliseconds at a specified speed.

— In Modkit, the duration quantity is a number that allows you to specify how many degrees or revolutions to move. Positive numbers spin the motor forward and negative numbers spin the motor backward.

**Key Differences**

In the VEX EDR Cortex system, the motors do not contain built-in encoders, so the Integrated Motor Encoders or Shaft Encoders must also be used with the code above. In the VEX EDR V5 system, all Smart Motors contain built-in encoders.

**VCS Commands Used:**

- `setVelocity()`
- `startRotateFor()`
- `waitUntil()`
- `wait()`
- `isDone()`
In the code below, the robot is programmed to move the arm motor for a specified amount of degrees, or encoder counts, then hold or brake the motor. The arm movement will time out after 5 seconds to prevent the program from locking up if the arm is unable to reach the target number of degrees. The top code is written in ROBOTC Graphical for a VEX EDR Cortex robot equipped with one motor attached arm with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in Modkit Block for a VEX EDR V5 robot equipped with one Smart Motor.

**Call Outs:**
Program start
In ROBOTC Graphical, the robot begins executing code at line 1.

In Modkit, the robot begins executing code at the top of the `when(STARTED)` event.

- **Move commands**
  - In ROBOTC Graphical, the `moveMotorTarget()` command allows you to turn on a specified motor for a specified number of encoder counts. The motor spins at the specified motor power level, slowing to a stop and then braking at the end of the movement when set to true.
  - In Modkit, the `rotateTo()` command allows you to turn on a specified motor to an absolute position in degrees or revolutions.

- **Specified motor**
  - In ROBOTC Graphical, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is the first argument of the `moveMotorTarget()` command.
  - In Modkit, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `rotateTo()` command.

- **Duration**
  - In ROBOTC Graphical, the duration within the `moveMotorTarget()` command is always a number of encoder counts from the Integrated Motor Encoder or Shaft Encoder.
  - In Modkit, the duration can be specified in degrees or revolutions for the specified Smart Motor.

- **Braking**
  - In ROBOTC Graphical, the motors can be set to brake or coast at the end of the movement. Setting the final argument of the `moveMotorTarget()` command to true sets the motors to brake at the end of the movement.
  - In Modkit, motors can be set to `hold`, `brake`, or `coast` using the `setStopping()` commands. All movements by a motor within the program will hold, brake or coast according to how it is set.

- **Stopping the motor**
  - In ROBOTC Graphical, the `wait()` command provides time for the robot to complete the movement, and prevents the program from locking up if that movement takes more than the specified amount of time.
  - In Modkit, the `setTimeout()` command is set on a specific motor. This ensures that the program will not lock up by limiting the amount of time the motor has to complete a movement.

- **Motor status**
— In ROBOTC Graphical, the `stopMotor()` stops the specified motor from spinning or trying to spin. This is used to ensure that the motor does not continuously try to move, even after it has reached its maximum encoder count limit.

— In Modkit, the `setTimeout()` command serves the function of stopping the specified motor after a specified amount of time.

**VCS Commands Used:**

- `setStopping()`
- `setTimeout()`
- `rotateTo()`
In the code below, the robot is programmed to move the claw motor for a specified amount of degrees, or encoder counts, then hold or brake the motor. The claw movement will time out after 3 seconds to prevent the program from locking up if the claw is unable to reach the target number of degrees. The top code is written in ROBOTC Graphical for a VEX EDR Cortex robot equipped with one motor attached to the claw with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in Modkit Block for a VEX EDR V5 robot equipped with one Smart Motor attached to a claw.

Call Outs:
Program start
— In ROBOTC Graphical, the robot begins executing code at line 1.
— In Modkit, the robot begins executing code at the top of the when(STARTED) event.

- Move commands
  — In ROBOTC Graphical, the moveMotorTarget() command allows you to turn on a specified motor for a specified number of encoder counts. The motor spins at the specified motor power level, slowing to a stop and then braking at the end of the movement when set to true.
  — In Modkit, the rotateTo() command allows you to turn on a specified motor to an absolute position in degrees or revolutions.

- Specified motor
  — In ROBOTC Graphical, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is the first argument of the moveMotorTarget() command.
  — In Modkit, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the rotateTo() command.

- Duration
  — In ROBOTC Graphical, the duration within the moveMotorTarget() command is always a number of encoder counts from the Integrated Motor Encoder or Shaft Encoder.
  — In Modkit, the duration can be specified in degrees or revolutions for the specified Smart Motor.

- Braking
  — In ROBOTC Graphical, the motors can be set to brake or coast at the end of the movement. Setting the final argument of the moveMotorTarget() command to true sets the motors to brake at the end of the movement.
  — In Modkit, motors can be set to hold, brake, or coast using the setStopping() commands. All movements by a motor within the program will hold, brake or coast according to how it is set.

- Stopping the motor
  — In ROBOTC Graphical, the wait() command provides time for the robot to complete the movement, and prevents the program from locking up if that movement takes more than the specified amount of time.
  — In Modkit, the setTimeout() command is set on a specific motor. This ensures that the program will not lock up by limiting the amount of time the motor has to complete a movement.

- Motor status
In ROBOTC Graphical, the `stopMotor()` stops the specified motor from spinning or trying to spin. This is used to ensure that the motor does not continuously try to move, even after it has reached its maximum encoder count limit.

In Modkit, the `setTimeout()` command serves the function of stopping the specified motor after a specified amount of time.

**VCS Commands Used:**

- `setStopping()`
- `setTimeout()`
- `rotateTo()`
In the code below, the robot is programmed to use the values from the Left and Right Joysticks to remotely control the Left and Right Motors, indefinitely. The top code is written in ROBOTC Graphical for a VEX EDR Cortex robot equipped with two driving motors and the VEXnet Joysticks. The bottom code is written in Modkit Block for a VEX EDR V5 robot equipped with two Smart Motors and the V5 Controller.

**Call Outs:**

**Program start**
- In ROBOTC Graphical, the robot begins executing code at line 1.
- In Modkit, the robot begins executing code at the top of the `when (STARTED)` event.

- **Loop**
— In ROBOTC Graphical, the `repeat(forever)` loop allows you to repeat the set of code between its curly braces forever.

— In Modkit, the `while()` loop allows you to repeat the set of code inside the block while the condition between the parenthesis is true.

- **Loop condition**

  — In ROBOTC Graphical, the condition between the parenthesis of the `repeat(forever)` loop enables the code to loop forever, or until the program is terminated.

  — In Modkit, the condition between the parenthesis of the `while()` loop determines whether the robot runs the code that belongs to the loop and repeats, or moves on in the program. A condition of `true` is always true, causing the loop to repeat indefinitely.

- **Specified motor**

  — In ROBOTC Graphical, the motors are built into the `tankControl()` command. The first parameter specifies which VEXnet Joystick axis controls the left driving motors and the second parameter specifies which axis controls the right driving motors. In this example, ch3 is assigned to the left motor, and ch2 is assigned to the right motor.

  — In Modkit, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `setVelocity()` command.

- **Setting the velocity**

  — In ROBOTC Graphical, the `tankControl()` command directly sets the values of the motors equal to the values from the VEXnet Joysticks.

  — In Modkit, you use the `setVelocity()` command in order to set a motor’s respective velocity equal to the joystick axis on the V5 Controller.

- **Setting the Controller Axis**

  — In ROBOTC Graphical, the first two parameters in the `tankControl()` command set which joystick axis is assigned to which motors. The first parameter is for assigning an axis to the left motor and the second parameter for the right motor.

  — In Modkit, the `Controller1.Axis#.position(percent)` block allows you to access the position values from a joystick axis, which can then be used with the `setVelocity()` command. The left and right motors must be set to different `Axis#` to have independent control.

- **Controller joystick values**

  — In ROBOTC Graphical, the `tankControl()` command automatically assigns position values from the joysticks to the motors, ranging from -127 to 127.

  — In Modkit, the `position(percent)` retrieves the values of the joysticks in a range of -100 to 100.

- **Motor Commands**
In ROBOTC Graphical, the motors are run at the desired speeds inside the tankControl() command.

In Modkit, the spin() command allows you to turn on the previously specified motor either forward or reverse, at a specified speed. The motor continues to run at this speed until the programmer changes it later in the code, or in this case, is updated with a new joystick value.

- **Motor direction**
  - In ROBOTC Graphical, this setting is already taken care of inside the tankControl() command.
  - In Modkit, the direction and speed in this program is continuously adjusted by setting the motors equal to values from the joysticks but is set to forward to begin with.

- **Threshold**
  - In ROBOTC Graphical, the third parameter of the tankControl() command determines the minimum joystick value required to move the robot.
  - In Modkit, there is no command block to set the threshold of the joysticks. If need to make a threshold value, you can create an algorithm using if-else statements.

**Key Differences**

In ROBOTC Graphical, the motors and VEXnet Joysticks use values ranging from -127 to +127. In Modkit, the Smart Motors and V5 Controller use percentage values ranging from -100 to 100. Another key difference is that for ROBOTC Graphical, assigning motors to VEXnet Joysticks can be done using just a repeat(forever) and tankControl() command, whereas in Modkit needs a while() loop, setVelocity(), position(), and spin() commands.

**VCS Commands Used:**

```c
while()
```

- setVelocity()
- spin()
- position()
Tank Driver Control with Buttons in ROBOTC Graphical and Modkit Block

In the code below, the robot is programmed to use the values from the Left and Right Joysticks to remotely control the Left and Right Motors, and the Top Right and Bottom Right Motors to control an Arm Motor, indefinitely. The top code is written in ROBOTC Graphical for a VEX EDR Cortex robot equipped with two driving motors, 1 arm motor, and the VEXnet Joysticks. The bottom code is written in Modkit Block for a VEX EDR V5 robot equipped with two driving Smart Motors, 1 Arm Smart Motor and the V5 Controller.

Call Outs:
Program start
  — In ROBOTC Graphical, the robot begins executing code at line 1.
  — In Modkit, the robot begins executing code at the top of the when(STARTED) event.
  • While and repeat( forever ) loop
— In ROBOTC Graphical, the `repeat(forever)` loop allows you to repeat the set of code between its curly braces forever.

— In Modkit, the `while()` loop allows you to repeat the set of code inside the block while the condition between the parenthesis is true.

• Loop condition
  — In ROBOTC Graphical, the condition between the parenthesis of the `repeat(forever)` loop enables the code to loop forever until the program is terminated.
  
  — In Modkit, the condition between the parenthesis of the `while()` loop determines whether the robot runs the code that belongs to the loop and repeats, or moves on in the program. A condition of `true` is always true, causing the loop to repeat indefinitely.

• Specified motor
  — In ROBOTC Graphical, the motors are built into the `tankControl()` command. The first two parameters represent the two driving motors and their assigned axis on the VEXnet Joysticks the move them. In this example, ch3 is assigned to the left motor, and ch2 is assigned to the right motor.

  — In Modkit, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `setVelocity()` command.

• Assigned motor
  — In ROBOTC Graphical, the `tankControl()` command assigns a motor's movement to joystick axis, and assigns a speed at which the motor will spin.

  — In Modkit, the `spin()` command allows you to turn on the previously specified motor either forward or backward, at a specified speed. The motor continues to run at this speed until the programmer changes it later in the code.

• If statements
  — In ROBOTC, the `armControl()` command internally uses if statements to detect if a button is pressed.

  — In Modkit, `if(){}` statements allow you to check a condition and then run any code between its curly braces. In this example, it is checking whether a button is pressed.

• Motor Commands
  — In ROBOTC Graphical, the `armControl()` command assigns a motor's movement to two buttons, and assigns a speed at which the motor will spin.

  — In Modkit, the `spin()` command allows you to turn on the previously specified motor either forward or reverse, at a specified speed. The motor continues to run at this speed until the programmer changes it later in the code. This program being inside the `if(Controller1.ButtonUp.pressing)` brackets assigns this motor and it's actions to the button.

• Else branch
— In ROBOTC Graphical, the armControl() command uses an else{} branch to stop spinning the assigned motor if neither of the two assigned buttons are pressed.

— In Modkit, the else{} branch allows you to run code when the condition of the previous, corresponding if{}{} statement is false.

- Arm motor stop

— In ROBOTC Graphical, the armControl() command internally tells the armMotor to stop spinning when the button is released.

— In Modkit, the stop{} command allows you to turn off the previously specified motor.

**Key Differences**

In ROBOTC Graphical, the motors and VEXnet Joysticks use values ranging from -127 to +127. In Modkit, the Smart Motors and V5 Controller use percentage values ranging from -100 to 100. The Smart Motors also allow you to set their brake type to hold, allowing the robot to maintain the position of an arm.

**VCS Commands Used:**

while()
- setVelocity()
- position()
- if{} else
- spin()
- stop()
Displaying Text to a Screen in ROBOTC and VEX C++

In the code below, the robot is programmed to display text to a screen. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with an LCD Display. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with just the Robot Brain.

ROBOTC

```c

1 task main()
{
    /*Display a string on the LCD Display starting at the first position on the screen */
    displayLCDString(0,0,"Hello World!");
}
```

VEX C++

```c

#include "robot-config.h"

int main() {
    //Displays text to the Robot Brain's screen
    Brain.Screen.print("Hello World!");
    Brain.Screen.printAt(1,40,"Hello Again World!");
}
```

Call Outs:

Main structure

- In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of task main.
- In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of int main.

- Display commands
— In ROBOTC, the displayLCDString() command allows you to display a string to the specified position on the LCD Display.

— In VEX C++, the Brain.Screen.print() command allows you print text (including strings, numbers, booleans) to the screen of the Robot Brain where the cursor currently is when the command is called. The Brain.Screen/printAt() command also displays text to the screen of the Robot Brain, but in the location specified by its parameters.

• Print location

— In ROBOTC, the first two parameters indicate which line to display on, and where on that line to start displaying. This is specified every time the command is called.

— In VEX C++, the Brain.Screen/printAt() command also uses its first two parameters to indicate where to display the text. However, instead of the parameters representing which line and position along the line to display text, they represent an X,Y pixel coordinate of where to start displaying the text.

• Text to display

— In ROBOTC, any strings within the parenthesis after the location parameters will display on the LCD Display. Numbers and chars will not display using this command.

— In VEX C++, using the Brain.Screen/print() command, any strings, numbers, or booleans within the parenthesis will display on the screen of the Robot Brain. If using the Brain.Screen/printAt() command, any strings, numbers, or booleans within the parenthesis after the location parameters will display on the screen of the Robot Brain.

Key Differences

One key difference between programming to display text to a screen in ROBOTC versus VEX C++ is that in ROBOTC, there are many different commands, depending if you want to display a string, char, or number. In VEX C++, there are only two commands to display text to the screen, and both commands can output strings, numbers, and booleans. The other key difference is that the print location parameters in ROBOTC commands are used to specify the line and position along that line, whereas in VEX C++, the print location parameters are for an X,Y pixel coordinate.

VCS Commands Used:

Brain.Screen/print()

• Brain.Screen/printAt()
In the code below, the robot is programmed to move forward a specified amount of time, and then stop. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with two driving motors. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with two Smart Motors.

**Call Outs:**

**Main structure**
- In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of task main.
- In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

**Move commands**
- In ROBOTC, the `motor[]` command allows you to turn on a specified motor at a specified power level. The motor continues to run at this power level until the
programmer changes it later in the code or the program ends. The direction and speed of the movement can be adjusted by modifying the power level.

— In VEX C++, the spin() command allows you to turn on the previously specified motor either forward or backward, at a specified speed. The motor continues to run at this speed until the programmer changes it later in the code. The movement can be adjusted by modifying the direction and speed.

- Specified motor
  — In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is specified in square brackets.
  — In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the spin() command.

- Speed
  — In ROBOTC, the speed (or motor power level) is specified in a range from -127 (full power reverse) to 127 (full power forward). A value of 63 is approximately half power.
  — In VEX C++, the speed is specified as a percent, dps or rpm.

- Time duration
  — In ROBOTC, the wait1Msec() command allows you to specify an amount of time in milliseconds to hold the program execution before moving on.
  — In VEX C++, the task::sleep() command allows you to specify an amount of time in milliseconds to hold the program execution before moving on.

- Stop commands
  — In ROBOTC, the motor[] command allows you to turn on a specified motor at a specified power level. The motor continues to run at this power level until the programmer changes it later in the code or the program ends. Setting a motor to 0 stops providing power to that motor.
  — In VEX C++, the stop() command allows you to turn off the previously specified motor. You are also able to specify whether you want the motor to brake, hold or coast at the end of the movement.

Key Differences

In the VEX EDR Cortex system the motors do not contain built-in encoders, so options to make the motor brake are not available by default. In the VEX EDR V5 system, all Smart Motors contain built-in encoders, allowing for options such as braking at the end of a movement.

VCS Commands Used:

spin()
- stop()
- task::sleep()
Moving Forward to a Target in ROBOTC and VEX C++

In the code below, the robot is programmed to move forward until it has reached a target number of encoder counts, and then stop. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with two driving motors with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with two Smart Motors.

Call Outs:

Main structure

- In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of task main.

- In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

- Move commands
In ROBOTC, the `moveMotorTarget()` command allows you to turn on a specified motor for a specified number of encoder counts. The motor spins at the specified motor power level, slowing to a stop and then braking at the end of the movement when set to true. The direction and speed of the movement can be adjusted by modifying the power level and encoder count.

In VEX C++, the `startRotateFor()` command allows you to turn on the previously specified motor for a specified duration (encoder counts or rotations) and a specified speed (percent or RPM). The direction and speed of the movement can be adjusted by modifying the speed and encoder count.

- **Specified motor**
  - In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is the first argument of the `moveMotorTarget()` command.
  - In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `startRotateFor()` command.

- **Duration**
  - In ROBOTC, the duration within the `moveMotorTarget()` command is always a number of encoder counts from the Integrated Motor Encoder or Shaft Encoder.
  - In VEX C++, the duration can be specified in encoder counts or revolutions of the Smart Motor.

- **Speed**
  - In ROBOTC, the speed (or motor power level) is specified in a range from -127 (full power reverse) to 127 (full power forward). A value of 63 is approximately half power.
  - In VEX C++, the speed is specified as a percent, dps, or rpm.

- **Braking**
  - In ROBOTC, the motors can be set to brake or coast at the end of the movement. Setting the final argument of the `moveMotorTarget()` command to true sets the motors to brake at the end of the movement.
  - In VEX C++, motors can be set to brake, hold or coast using the `setStopping()` commands. All movements by a motor within the program will brake or coast according to how it is set.

- **The while loop**
  - In ROBOTC, the `moveMotorTarget()` commands do not wait for the movement to complete before program execution continues. A while loop is used to keep the program execution from moving on until both motors have complete their movements.
  - In VEX C++, the `startRotateFor()` commands do not wait for the movement to complete before program execution continues. A while loop is used to keep the program execution from moving on until both motors have complete their movements.

- **Motor status**
— In ROBOTC, the status of a motor (whether it has finished spinning) can be checked using the `getMotorTargetCompleted()` command.

— In VEX C++, the status of a motor (whether it is spinning or not) can be checked using the `isSpinning()` command.

**Key Differences**

In the VEX EDR Cortex system, the motors do not contain built-in encoders, so the Integrated Motor Encoders or Shaft Encoders must also be used with the code above. In the VEX EDR V5 system, all Smart Motors contain built-in encoders.

**VCS Commands Used:**

- `setStopping()`
- `startRotateFor()`
- `isSpinning()`
In the code below, the robot is programmed to turn to the left for a specified amount of time, and then stop. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with two driving motors. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with two Smart Motors.

**Call Outs:**

**Main structure**
- In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of `task main`.
- In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

**Move commands**
- In ROBOTC, the `motor []` command allows you to turn on a specified motor at a specified power level. The motor continues to run at this power level until the
programmer changes it later in the code or the program ends. The direction and speed of the movement can be adjusted by modifying the power level.

— In VEX C++, the `spin()` command allows you to turn on the previously specified motor either forward or in reverse, at a specified speed. Specifying `directionType::rev` will cause the motor to spin in reverse. The motor continues to run at the specified speed and direction until the programmer changes it later in the code.

• Specified motor
  — In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is specified in square brackets.
  — In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `spin()` command.

• Speed
  — In ROBOTC, the speed (or motor power level) is specified in a range from -127 (full power reverse) to 127 (full power forward). A value of -63 is approximately half power, in reverse.
  — In VEX C++, the speed is specified as a percent, dps, or rpm.

• Time duration
  — In ROBOTC, the `wait1Msec()` command allows you to specify an amount of time in milliseconds to hold the program execution before moving on.
  — In VEX C++, the `task::sleep()` command allows you to specify an amount of time in milliseconds to hold the program execution before moving on.

• Stop commands
  — In ROBOTC, the `motor[]` command allows you to turn on a specified motor at a specified power level. The motor continues to run at this power level until the programmer changes it later in the code or the program ends. Setting a motor to 0 stops providing power to that motor.
  — In VEX C++, the `stop()` command allows you to turn off the previously specified motor. You are also able to specify whether you want the motor to brake, hold, or coast at the end of the movement.

**Key Differences**

In the VEX EDR Cortex system the motors do not contain built-in encoders, so options to make the motor brake are not available by default. In the VEX EDR V5 system, all Smart Motors contain built-in encoders, allowing for options such as braking at the end of a movement.
VCS Commands Used:

spin()
- stop()
- task::sleep()
Turning to a Target in ROBOTC and VEX C++

In the code below, the robot is programmed to turn left until it has reached a target number of encoder counts, and then stop. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with two driving motors with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with two Smart Motors.

Call Outs:

Main structure

— In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of `task main`.

— In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

• Move commands
— In ROBOTC, the `moveMotorTarget()` command allows you to turn on a specified motor for a specified number of encoder counts. The motor spins at the specified motor power level, slowing to a stop and then braking at the end of the movement when set to true. The direction and speed of the movement can be adjusted by modifying the power level and encoder count.

— In VEX C++, the `startRotateFor()` command allows you to turn on the previously specified motor for a specified duration (encoder counts or rotations) and a specified speed (percent or RPM). The direction and speed of the movement can be adjusted by modifying the speed and encoder count.

- **Specified motor**
  
  — In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is the first argument of the `moveMotorTarget()` command.
  
  — In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `startRotateFor()` command.

- **Duration**
  
  — In ROBOTC, the duration within the `moveMotorTarget()` command is always a number of encoder counts from the Integrated Motor Encoder or Shaft Encoder. Setting the encoder counts to negative will cause the motor to spin in reverse.
  
  — In VEX C++, the duration can be specified in encoder counts or rotations of the Smart Motor. Setting a negative encoder count or rotation will cause the motor to spin in reverse.

- **Speed**
  
  — In ROBOTC, the speed (or motor power level) is specified in a range from -127 (full power reverse) to 127 (full power forward). A value of -63 is approximately half power, in reverse.
  
  — In VEX C++, the speed is specified as a percent, dps, or rpm.

- **Braking**
  
  — In ROBOTC, the motors can be set to brake or coast at the end of the movement. Setting the final argument of the `moveMotorTarget()` command to true sets the motors to brake at the end of the movement.
  
  — In VEX C++, motors can be set to brake, hold, or coast using the `setStopping()` commands. All movements by a motor within the program will brake or coast according to how it is set.

- **The while loop**
  
  — In ROBOTC, the `moveMotorTarget()` commands do not wait for the movement to complete before program execution continues. A while loop is used to keep the program execution from moving on until both motors have complete their movements.
  
  — In VEX C++, the `startRotateFor()` commands do not wait for the movement to complete before program execution continues. The conditions in the while loop is
used to prevent the program execution from moving on until both motors have completed their movements.

- Motor status
  
  - In ROBOTC, the status of a motor (whether it has finished spinning) can be checked using the getMotorTargetCompleted() command.
  
  - In VEX C++, the status of a motor (whether it has finished spinning) can be checked using the isSpinning() command.

**Key Differences**

In the VEX EDR Cortex system, the motors do not contain built-in encoders, so the Integrated Motor Encoders or Shaft Encoders must also be used with the code above. In the VEX EDR V5 system, all Smart Motors contain built-in encoders.

**VCS Commands Used:**

- setStopping()
- startRotateFor()
- isSpinning()
Programming the Robot Arm in ROBOTC and VEX C++

In the code below, the robot is programmed to move the arm motor for a specified number of degrees, or encoder counts, then hold or brake the motor. The arm movement will time out after 5 seconds to prevent the program from locking up if the arm is unable to reach the target number of degrees. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with one motor attached to an arm with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with one Smart Motor.

```
1 task main()
2 {
3     //Start the arm motor spinning and brake when done
4     moveMotorTarget(armMotor, 630, 50, true);
5     //Wait 5 seconds
6     waitMsec(5000);
7     //Stop the motor from spinning
8     motor[armMotor]=0;
9 }
```

```
#include "robot-config.h"

int main() {
1     //Set the brake mode of the arm motor
2     ArmMotor.setStopping(brakeType::hold);
3     ArmMotor.setBrake(brakeType::hold);
4     //Sets a countdown to 5 seconds for the claw motor to wait before timing out
5     ArmMotor.setTimeout(5, timeUnits::sec);
6     //Stops the claw motor spinning
7     ArmMotor.rotateTo(630, rotationUnits::deg, 50, velocityUnits::pct);
8 }
```

**Call Outs:**

Main structure
— In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of `task main`.

— In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

**Move commands**

— In ROBOTC, the `moveMotorTarget()` command allows you to turn on a specified motor for a specified number of encoder counts. The motor spins at the specified motor power level, slowing to a stop and then braking at the end of the movement when set to true.

— In VEX C++, the `rotateTo()` command allows you to turn on a specified motor for a specified number of rotations (in degree, radians, or raw encoder values) and a specified speed (percent or RPM).

**Specified motor**

— In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is the first argument of the `moveMotorTarget()` command.

— In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `rotateTo()` command.

**Duration**

— In ROBOTC, the duration within the `moveMotorTarget()` command is always a number of encoder counts from the Integrated Motor Encoder or Shaft Encoder.

— In VEX C++, the duration can be specified in degree, radians, or raw encoder values for the specified Smart Motor.

**Speed**

— In ROBOTC, the speed (or motor power level) is specified in a range from -127 (full power reverse) to 127 (full power forward).

— In VEX C++, the speed is specified as a percent, rotations per minute, or degrees per second.

**Braking**

— In ROBOTC, the motors can be set to brake or coast at the end of the movement. Setting the final argument of the `moveMotorTarget()` command to `true` sets the motors to brake at the end of the movement.

— In VEX C++, motors can be set to `hold`, `brake`, or `coast` using the `setStopping()` commands. All movements by a motor within the program will hold, brake or coast according to how it is set.

**Stopping the motor**
In ROBOTC, the wait1Msec() command provides time for the robot to complete the arm movement, and prevents the program from locking up if that movement takes more than the specified amount of time.

In VEX C++, the setTimeout() command is set on a specific motor. This ensures that the program will not lock up by limiting the amount of time the motor has to complete a movement.

- Motor status

  - In ROBOTC, the motor[] = 0 stops the specified motor from spinning or trying to spin. This is used to ensure that the motor does not continuously try to move, even after it has reached its maximum encoder count limit.

  - In VEX C++, the setTimeout() command serves the function of stopping the specified motor after the specified amount of time.

**VCS Commands Used:**

- setStopping()
- setTimeout()
- rotateTo()
In the code below, the robot is programmed to move the claw motor for a specified amount of degrees, or encoder counts, then hold or brake the motor. The claw movement will time out after 3 seconds to prevent the program from locking up if the claw is unable to reach the target number of degrees. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with one motor attached to the claw with Integrated Motor Encoders or Shaft Encoders. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with one Smart Motor attached to a claw.

```c
// Start the claw motor spinning and brake when done
moveMotorTarget(clawMotor, 570, 50, true);

// Wait 3 seconds
waitMsec(3000);

// Stop the motor from spinning
motor[clawMotor]=0;

#include "robot-config.h"

int main() {

// Set the brake mode of the claw motor
ClawMotor.setStopping(brakeType::hold);

// Sets a countdown to 3 seconds for the claw motor to wait before timing out
ClawMotor.setTimeout(3, timeUnits::sec);

// Starts the claw motor spinning
ClawMotor.rotateTo(570, rotationUnits::deg, 50, velocityUnits::pct);
}
```

**Call Outs:**

Main structure
— In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of `task main`.

— In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

- **Move commands**
  - In ROBOTC, the `moveMotorTarget()` command allows you to turn on a specified motor for a specified number of encoder counts. The motor spins at the specified motor power level, slowing to a stop and then braking at the end of the movement when set to true.
  
  - In VEX C++, the `rotateTo()` command allows you to turn on a specified motor to an absolute position (in degree, radians, or raw encoder values) and a specified speed (percent, dps, or rpm).

- **Specified motor**
  - In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is the first argument of the `moveMotorTarget()` command.

  - In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the `rotateTo()` command.

- **Duration**
  - In ROBOTC, the duration within the `moveMotorTarget()` command is always a number of encoder counts from the Integrated Motor Encoder or Shaft Encoder.

  - In VEX C++, the duration can be specified in degree, radians, or raw encoder values for the specified Smart Motor.

- **Speed**
  - In ROBOTC, the speed (or motor power level) is specified in a range from -127 (full power reverse) to 127 (full power forward). A value of 63 is approximately half power.

  - In VEX C++, the speed is specified as a percent, rotations per minute, or degrees per second.

- **Braking**
  - In ROBOTC, the motors can be set to brake or coast at the end of the movement. Setting the final argument of the `moveMotorTarget()` command to true sets the motors to brake at the end of the movement.

  - In VEX C++, motors can be set to hold, brake, or coast using the `setStopping()` commands. All movements by a motor within the program will hold, brake or coast according to how it is set.

- **Stopping the motor**
In ROBOTC, the `wait1Msec()` command provides time for the robot to complete the claw movement, and prevents the program from locking up if that movement takes more than the specified amount of time.

In VEX C++, the `setTimeout()` command is set on a specific motor. This ensures that the program will not lock up by limiting the amount of time the motor has to complete a movement.

- **Motor status**

  In ROBOTC, the `motor[] = 0` stops the specified motor from spinning or trying to spin. This is used to ensure that the motor does not continuously try to move, even after it has reached its maximum encoder count limit.

  In VEX C++, the `setTimeout()` command serves the function of stopping the specified motor after a specified amount of time.

**VCS Commands Used:**

- `setStopping()`
- `setTimeout()`
- `rotateTo()`
Tank Driver Control in ROBOTC and VEX C++

In the code below, the robot is programmed to use the values from the Left and Right Joysticks to remotely control the Left and Right Motors, indefinitely. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with two driving motors and the VEXnet Joysticks. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with two Smart Motors and the V5 Controller.

```
#include "robot-config.h"

int main() {
    while(true) {
        //Set the Left and Right Motors equal to the position values of the Left and Right Joysticks
        RightMotor.spin(directionType::fwd,Controller1.Axis2.position(percentUnits::pct),velocityUnits::pct);
        LeftMotor.spin(directionType::fwd,Controller1.Axis3.position(percentUnits::pct),velocityUnits::pct);
    }
}
```

**Call Outs:**

**Main structure**

- In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of task main.
- In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of `int main`.

- **While loop**
— In ROBOTC, the while() loop allows you to repeat the set of code between its curly braces while the condition between the parenthesis is true.

— In VEX C++, the while() loop also allows you to repeat the set of code between its curly braces while the condition between the parenthesis is true.

• Loop condition

— In ROBOTC, the condition between the parenthesis of the while() loop determines whether the robot runs the code that belongs to the loop and repeats, or moves on in the program. A condition of 1==1 is always true, causing the loop to repeat indefinitely. A condition of true would also be accepted in ROBOTC.

— In VEX C++, the condition between the parenthesis of the while() loop determines whether the robot runs the code that belongs to the loop and repeats, or moves on in the program. A condition of true is always true, causing the loop to repeat indefinitely. A condition of 1==1 would also be accepted in VEX C++.

• Specified motor

— In ROBOTC, the name of the motor is specified in the Motors and Sensors Setup. The name of the motor is specified in square brackets.

— In VEX C++, the name of the motor is specified in the Robot configuration screen. The name of the motor is specified before the spin() command.

• Motor commands

— In ROBOTC, the motor[] command allows you to turn on the specified motor at a specified motor power level. The motor continues to run at that motor power level until the programmer changes it later in the code or the program ends. In this program, the motor power level is continuously adjusted by setting the motors equal to values from the joysticks.

— In VEX C++, the spin() command allows you to turn on the previously specified motor either forward or backward, at a specified speed. The motor continues to run at this speed until the programmer changes it later in the code or the program ends. In this program, the direction and speed is continuously adjusted by setting the motors equal to values from the joysticks.

• Controller commands

— In ROBOTC, the vexRT[] command allows you to access a value from the VEXnet Joysticks. The joystick axis or button is specified within the square brackets of the command.

— In VEX C++, the position() command allows you to access a position value from the previously specified joystick axis on the V5 Controller. The value() command can also work here.

• Controller joystick values
— In ROBOTC, the Ch# keywords allows you to access the position values from one of the joystick axis on the VEXnet Joysticks.
— In VEX C++, the Controller1.Axis# keywords allow you to access the position values from one of the joystick axis on the V5 Controller.

Key Differences

In ROBOTC, the motors and VEXnet Joysticks use values ranging from -127 to +127. In VEX C++, the Smart Motors and V5 Controller use percentage values ranging from -100 to 100.

VCS Commands Used:

spin()

• position()
Tank Driver Control with Arm in ROBOTC and VEX C++

In the code below, the robot is programmed to use the values from the Left and Right Joysticks to remotely control the Left and Right Motors, and the Top Right and Bottom Right Motors to control an Arm Motor, indefinitely. The top code is written in ROBOTC for a VEX EDR Cortex robot equipped with two driving motors, 1 arm motor, and the VEXnet Joysticks. The bottom code is written in VEX C++ for a VEX EDR V5 robot equipped with two driving Smart Motors, 1 Arm Smart Motor and the V5 Controller.

```cpp
#1  task main()
#2  {
#3      while(1 == 1){
#4          //Set the Left and Right Motors equal to the position values of the Left and Right Joysticks
#5          motor[leftMotor] = vexRT[Ch3];
#6          motor[rightMotor] = vexRT[Ch2];
#7          //Raise the Arm Motor if the R1 Button is pressed
#8          if(vexRT[BtnR1] == 1){
#9              motor[armMotor] = 50;
#10             //Else lower the Arm Motor if the R2 Button is pressed
#11          } else if(vexRT[BtnR2] == 1){
#12              motor[armMotor] = -50;
#13          } //Else stop the arm and hold its current position
#14          else{
#15              motor[armMotor] = 0;
#16          }
#17      }
#18  }
```
Call Outs:

Main structure

— In ROBOTC, `task main()` serves as the primary structure of the program. A robot begins executing code at the beginning of task main.

— In VEX C++, `int main()` serves as the primary structure of the program. A robot begins executing at the beginning of int main.

• While loop

— In ROBOTC, the `while()` loop allows you to repeat the set of code between its curly braces while the condition between the parenthesis is true.

— In VEX C++, the `while()` loop also allows you to repeat the set of code between its curly braces while the condition between the parenthesis is true.

• Loop condition

— In ROBOTC, the condition between the parenthesis of the `while()` loop determines whether the robot runs the code that belongs to the loop and repeats, or moves on in the program. A condition of `1==1` is always true, causing the loop to repeat indefinitely. A condition of `true` would also be accepted in ROBOTC.

— In VEX C++, the condition between the parenthesis of the `while()` loop determines whether the robot runs the code that belongs to the loop and repeats, or moves on in the program. A condition of `true` is always true, causing the loop to repeat indefinitely. A condition of `1==1` would also be accepted in VEX C++.

• Tank Drive
— In ROBOTC, the `motor[]` command allows you to turn on the specified motor at a specified motor power level. In this program, the motor power level is continuously adjusted by setting the motors equal to values from the joysticks. The `vexRT[]` command allows you to access a value from the VEXnet Joysticks. The joystick axis or button is specified within the square brackets of the command.

— In VEX C++, the `spin()` command allows you to turn on the previously specified motor either forward or backward, at a specified speed. In this program, the direction and speed is continuously adjusted by setting the motors equal to values from the joysticks. The `position()` command allows you to access a position value from the previously specified joystick axis on the V5 Controller.

- **If statements**
  — In ROBOTC, `if(){}` statements allow you to check a condition, such as whether a button is pressed, and then run any code between its curly braces.
  — In VEX C++, `if(){}` statements also allow you to check a condition, such as whether a button is pressed, and then run any code between its curly braces.

- **Check for button press**
  — In ROBOTC, the `vexRT[]` command allows you to access a button or joystick value from the VEXnet Joysticks. The joystick axis or button is specified within the square brackets of the command. Checking if a button is equal to one (`== 1`) is checking if the specified button is pressed.
  — In VEX C++, the `pressing()` command allows you to check if the previously specified button on the V5 Controller is pressed or released. It has a value of true when the button is pressed, and false when released.

- **Motor commands**
  — In ROBOTC, the `motor[]` command allows you to turn on a specified motor at a specified power level. The motor continues to run at this power level until the programmer changes it later in the code or the program ends.
  — In VEX C++, the `spin()` command allows you to turn on the previously specified motor either forward or backward, at a specified speed. The motor continues to run at this speed until the programmer changes it later in the code or the program ends.

- **Else branch**
  — In ROBOTC, the `else{}` branch allows you to run code when the condition of the previous, corresponding `if(){}` statement is false.
  — In VEX C++, the `else{}` branch also allows you to run code when the condition of the previous, corresponding `if(){}` statement is false.

- **Arm motor stop**
— In ROBOTC, the `motor[]` command allows you to turn off a specified motor by setting it equal to 0.

— In VEX C++, the `stop()` command allows you to turn off the previously specified motor. It also allows you to choose if you want the previously specified motor to brake, coast, or hold once it is stopped. Setting `brakeType::hold` will cause the motor to continue to draw power to maintain the current position of the arm.

**Key Differences**

In ROBOTC, the motors and VEXnet Joysticks use values ranging from -127 to +127. In VEX C++, the Smart Motors and V5 Controller use percentage values ranging from -100 to 100. The Smart Motors also allow you to set their brake mode to hold, allowing the robot to maintain the position of an arm.

**VCS Commands Used:**

- `spin()`
- `position()`
- `pressing()`
- `stop()`
Week 1

In this session, we will take a look at some of the new VEX V5 system hardware. Understanding the enhancements to components like the V5 Brain and Smart Motors is foundational to understanding some of the changes in the new programming environment, VEX Coding Studio. We will also go over procedures such as updating the firmware, configuring Smart Motors, configuring sensors, and downloading and running a basic VEX C++ Program.

Watch the Week 1 recording: https://youtu.be/9-enZx6ePII

Week 2

In this session, we will spend more time understanding how movement programs in ROBOTC text translate to similar movement programs with VEX C++. We will cover behaviors including Moving for Time, Moving to an Encoder Target, Arm Control, and Claw Control.

Watch the Week 2 recording: https://youtu.be/861C3AYXfs

Week 3

In this session, we will deepen our understanding of VEX C++ programming in VEX Coding Studio. We will cover additional behaviors including Displaying Text, Remote Control, and Moving until Sensor Values.
Watch the Week 3 recording: https://youtu.be/jRSuI8Yr32A

**Week 4**

In this session, we will take a look at how your experience with ROBOTC Graphical translates to Modkit Blocks in VEX Coding Studio. Although both languages are block based, Modkit introduces the concept of Event-Based Programming, which we will spend some time discussing. We will cover behaviors around Movement, Displaying Text, and Remote Control.

Watch the Week 4 recording: https://youtu.be/mv9RLLei4DM