```
#pragma config(Sensor, in1,    bottomPOT,      sensorPotentiometer)
#pragma config(Sensor, in2,    topPOT,         sensorPotentiometer)
#pragma config(Sensor, dgtl2,  backRightENC,   sensorQuadEncoder)
#pragma config(Sensor, dgtl4,  backLeftENC,    sensorQuadEncoder)
#pragma config(Motor,  port2,           rightBack,     tmotorServoContinuousRota
#pragma config(Motor,  port3,           rightFront,    tmotorServoContinuousRota
#pragma config(Motor,  port5,           bottomLift,    tmotorServoContinuousRota
#pragma config(Motor,  port6,           topLift,       tmotorServoContinuousRota
#pragma config(Motor,  port7,           claw,          tmotorServoContinuousRota
#pragma config(Motor,  port8,           leftBack,      tmotorServoContinuousRota
#pragma config(Motor,  port9,           leftFront,     tmotorServoContinuousRota
//*!!Code automatically generated by 'ROBOTC' configuration wizard

/*---------------------------------------------------------------------------*/
/*                                                                           */
/*        Description: Competition template for VEX EDR                      */
/*                                                                           */
/*---------------------------------------------------------------------------*/

// This code is for the VEX cortex platform
#pragma platform(VEX2)

// Select Download mthod as "competition"
#pragma competitionControl(Competition)

//Main competition background code...do not modify!
#include "Vex_Competition_Includes.c"

///SmartMotorLibrary by JPearman on the VEX forums
#include "Libraries/SmartMotorLib.c"

#pragma systemFile

/*---------------------------------------------------------------------------*/
/*                           Pre-Autonomous Functions                        */
/*                                                                           */
/*  You may want to perform some actions before the competition starts.      */
/*  Do them in the following function.  You must return from this function   */
/*  or the autonomous and usercontrol tasks will not be started.  This       */
/*  function is only called once after the cortex has been powered on and    */
/*  not every time that the robot is disabled.                               */
/*---------------------------------------------------------------------------*/

void DriveForTime (int power, int time) {
  SetMotor(leftFront, power, false);
  SetMotor(rightFront, power, false);
  SetMotor(leftBack, power, false);
  SetMotor(rightBack, power, false);

  wait1Msec(time);

  SetMotor(leftFront, 0, false);
  SetMotor(rightFront, 0, false);
  SetMotor(leftBack, 0, false);
  SetMotor(rightBack, 0, false);

}

/*void waitUntilQuadrature(int sensorChosen, int amountToGo){
```

```
  int currentCount = SensorValue[sensorChosen];
  int finalAmount = currentCount + amountToGo;

  while( SensorValue[sensorChosen] <= finalAmount ) { wait1Msec(10);}
}*/


void DriveForClicks(int encChosen, int amountToGo, int power){

  // get initial encoder clicks; calculate ending click value
  // function uses relative values for encoder clicks instead of
  // resetting enc to 0
  int currentCount = SensorValue[encChosen];
  int finalAmount = currentCount + amountToGo;

  // turn on motors to desired power
  SetMotor(rightBack, power, false);
  SetMotor (rightFront, power, false);
  SetMotor (leftBack, power, false);
  SetMotor (leftFront, power, false);

  // keep checking sensor value with tiny wait
  // code will stay in this statement until clicks reached
  while(SensorValue[encChosen] <= finalAmount) {wait1Msec(20);}

  SetMotor(rightBack, 0, false);
  SetMotor (rightFront, 0, false);
  SetMotor (leftBack, 0, false);
  SetMotor (leftFront, 0, false);
}

/*void changeClaw (int direction){
  if (direction == 1) { // 1 = open
    SetMotor (claw,80,false);
  }

  else {
    SetMotor (claw,-80,false);
  }
  wait1Msec(800);
  if (direction == 1) { // 1 = open
    SetMotor (claw,10,false);
  }

  else {
    SetMotor (claw,-10,false);
  }
}*/


// use an asterisk on direction variable below; makes it a pointer
// this is the way you have to pass strings/characters to a function in RobotC
void liftUsingPOT (int power, int topPOTlimit, int bottomPOTlimit, string *direc

  // get initial sensor values
  int topPOTvalue = SensorValue[topPOT];
  int bottomPOTvalue = SensorValue[bottomPOT];

  // turn on motors to desired power
```

```
  SetMotor(bottomLift, power, false);
  SetMotor(topLift, power, false);

  if ( *direction == "up" ) {

    // run this loop while both POTs are under our set limits
    while (topPOTvalue <= topPOTlimit && bottomPOTvalue <= bottomPOTlimit){

      //keep checking potentiometer values
      topPOTvalue = SensorValue[topPOT];
      bottomPOTvalue = SensorValue[bottomPOT];

      // add a small wait; it doesn't pay to keep checking the
      // sensor value all the time
      wait1Msec(50);
    }
  }

  // otherwise, the direction must be "down"
  else {

    // run this loop while both POTs are over our set limits
    while (topPOTvalue >= topPOTlimit && bottomPOTvalue >= bottomPOTlimit){

      //keep checking potentiometer values
      topPOTvalue = SensorValue[topPOT];
      bottomPOTvalue = SensorValue[bottomPOT];

      // add a small wait here; it doesn't pay to keep checking the
      // sensor value continuously
      wait1Msec(50);
    }
  }

  // after POT limit is hit, turn off motors
  SetMotor(bottomLift, 0, false);
  SetMotor(topLift, 0, false);
}

/*void lowerPOT (int power, int topPOTdest, int bottomPOTdest){
  int runloop=1;
  int topPOTvalue;
  int bottomPOTvalue;
  while (runloop==1){
    //potentiometer values
    topPOTvalue = SensorValue[topPOT];
    bottomPOTvalue = SensorValue[bottomPOT];
    if (topPOTvalue <= topPOTdest || bottomPOTvalue <= bottomPOTdest){
      {
        power = 0;
        runloop=0;
      }
    }
    SetMotor(bottomLift, power, false);
    SetMotor(topLift, power, false);
  }
}
*/
```

```
void pre_auton()
{

  SmartMotorsInit();
  // Set bStopTasksBetweenModes to false if you want to keep user created tasks
  // running between Autonomous and Driver controlled modes. You will need to
  // manage all user created tasks if set to false.
  bStopTasksBetweenModes = true;

  // Set bDisplayCompetitionStatusOnLcd to false if you don't want the LCD
  // used by the competition include file, for example, you might want
  // to display your team name on the LCD in this function.
  // bDisplayCompetitionStatusOnLcd = false;

  // All activities that occur before the competition starts
  // Example: clearing encoders, setting servo positions, ...
}

/*---------------------------------------------------------------------------*/
/*                                                                           */
/*                              Autonomous Task                              */
/*                                                                           */
/*  This task is used to control your robot during the autonomous phase of   */
/*  a VEX Competition.                                                       */
/*                                                                           */
/*---------------------------------------------------------------------------*/

task autonomous()
{
  SmartMotorRun();
  SmartMotorPtcMonitorEnable ();

  // keep this wait statement here;
  // program does not seem to run the first
  // motor command without it
  wait1Msec(100);


  // Drive forward 70 power / 500ms
  // pushes cone forward so claw will be able to reach it
  DriveForTime (70, 500);

  wait1Msec(500);

  // Drive backward -70 power / 427ms
  // moves backward to give claw some grabbing room
  DriveForTime (-70, 427);

  wait1Msec(500);

  //Close Claw to pop it out from folded position
  SetMotor(claw, -80, false);
  wait1Msec(500);
  SetMotor(claw, 0, false);

  // small wait to avoid stressing motors, going from
  // close direction to open direction
  wait1Msec(100);
```

```
  //Open Claw after it's flipped out
  SetMotor(claw, 127, false);
  wait1Msec(250);
  // apply just enough power to hold the claw open
  // but not so much that it stalls the motor
  SetMotor(claw, 30, false);

  // Drive forward, 70 power / 270ms
  // moves back to where cone is
  DriveForTime (70, 270);

  wait1Msec(500);

  //Close Claw to grab cone
  SetMotor(claw, -80, false);
  wait1Msec(500);
  // apply a small amount of power to make sure claw
  // stays closed, but not so much it will stall.
  // need less power here because rubber bands help
  // keep claw closed
  SetMotor(claw, -20, false);

  //Raise lift: 127 power / bottomLimit 1500 / topLimit 1400
  string liftDirection = "up";
  liftUsingPOT (127, 1500, 1400, liftDirection);
  wait1Msec(1000);

  // Drive foward to tower
  // use back left encoder to measure / 220 clicks / 100 power
  DriveForClicks(backLeftENC, 220, 100);
  wait1Msec(500);

  //Lower lift: -90 power, bottomPOT limit 1300 / topPOT limit 1200
  liftDirection = "down";
  liftUsingPOT(-90, 1300, 1200, liftDirection);

  //Open claw to release cone
  SetMotor(claw, -80, false);
  wait1Msec(500);
  SetMotor(claw, 0, false);

  // Drive backward, -127 power / 300ms
  // to get away from tower
  DriveForTime (-127, 300);

}

/*---------------------------------------------------------------------------*/
/*                                                                           */
/*                              User Control Task                            */
/*                                                                           */
/*  This task is used to control your robot during the user control phase of */
/*  a VEX Competition.                                                       */
/*                                                                           */
/*---------------------------------------------------------------------------*/


task usercontrol(){
```

```
  SmartMotorRun();
  SmartMotorPtcMonitorEnable();

  // chassis variables -------
  int rightpower = 0;
  int leftpower = 0;
  int adjLeft = 0;
  int adjRight = 0;
  int inverseBtn = 0;

  // claw variables -------
  int clawOpen = 0;
  int clawClose = 0;

  // lift variables --------
  int topPOTvalue;
  int bottomPOTvalue;

  int maxPOTtop = 2222;
  int maxPOTbottom = 2176;

  int topPower = 0;
  int bottomPower = 0;


#define MAX_POWER 127
#define DEADBAND 5

  while (true){
    /////////// Chassis //////////
    leftpower = (vexRT[Ch3] + vexRT[Ch4]);
    rightpower = (vexRT[Ch3] - vexRT[Ch4]);

    if(abs(rightpower) > MAX_POWER){
      rightpower = sgn(rightpower) * MAX_POWER;
    }

    if(abs(leftpower) > MAX_POWER){
      leftpower = sgn(leftpower) * MAX_POWER;
    }

    if(abs(leftpower) < DEADBAND) leftpower = 0;
    if(abs(rightpower) < DEADBAND) rightpower = 0;

    // make the back of the robot the front
    // by pressing Button 6U
    inverseBtn = vexRT(Btn6U);
    if (inverseBtn == 1){
      leftpower = leftpower*-1;
      rightpower = rightpower*-1;
    }

    // set left side motors;
    SetMotor(leftFront, leftpower, false);
    SetMotor(leftBack, leftpower, false);

    // set right side motors;
    SetMotor(rightFront, rightpower, false);
    SetMotor(rightBack, rightpower, false);
```

```
    // small adjust
    adjLeft = vexRT[Btn7L];
    adjRight = vexRT[Btn7R];

    if (adjLeft == 1){
      SetMotor(rightFront, 70, false);
      SetMotor(rightBack, 70, false);
      SetMotor(leftFront, 0, false);
      SetMotor(leftBack, 0, false);
    }
    else if(adjRight == 1) {
      SetMotor(leftFront, 70, false);
      SetMotor(leftBack, 70, false);
      SetMotor(rightFront, 0, false);
      SetMotor(rightBack, 0, false);
    }


    /////////// Claw //////////
    clawOpen = vexRT[Btn6UXmtr2];
    clawClose = vexRT[Btn6DXmtr2];

    if(clawClose == 1) SetMotor(claw, -100, false);
    else if( clawOpen == 1 ) SetMotor(claw, 100, false);
    else SetMotor(claw, 0, false);


    /////////// Lift //////////

    //potentiometer values
    topPOTvalue = SensorValue[topPOT];
    bottomPOTvalue = SensorValue[bottomPOT];


    // lift to joystick
    topPower = vexRT[Ch3Xmtr2];
    bottomPower = vexRT[Ch3Xmtr2];

    if (topPOTvalue >= maxPOTtop || bottomPOTvalue >= maxPOTbottom){
      if (vexRT[Ch3Xmtr2] > 0 ){
        topPower = 0;
        bottomPower = 0;
      }
    }
    SetMotor(bottomLift, bottomPower, false);
    SetMotor(topLift, topPower, false);

  }
}
```