```
#pragma config(UART Usage, UART1, uartVEXLCD, baudRate19200, IOPins, None, None)
#pragma config(UART Usage, UART2, uartNotUsed, baudRate4800, IOPins, None, None)
#pragma config(Sensor, in1,    leftLift,       sensorNone)
#pragma config(Sensor, in2,    rightLift,      sensorPotentiometer)
#pragma config(Sensor, in3,    rotatorPot,     sensorPotentiometer)
#pragma config(Sensor, in4,    clawPot,        sensorPotentiometer)
#pragma config(Sensor, in5,    gyro,           sensorGyro)
#pragma config(Sensor, dgtl8,  shooterLimit,   sensorDigitalIn)
#pragma config(Sensor, dgtl9,  leftDriveQuad,  sensorQuadEncoder)
#pragma config(Sensor, dgtl11, rightDriveQuad, sensorQuadEncoder)
#pragma config(Motor,  port2,           topLift,       tmotorVex393_MC29, openLo
#pragma config(Motor,  port3,           slingshot,     tmotorVex393_MC29, openLo
#pragma config(Motor,  port4,           frontRightDrive, tmotorVex393_MC29, oper
#pragma config(Motor,  port5,           frontLeftDrive, tmotorVex393_MC29, openl
#pragma config(Motor,  port6,           backRightDrive, tmotorVex393_MC29, openl
#pragma config(Motor,  port7,           backLeftDrive, tmotorVex393_MC29, openLo
#pragma config(Motor,  port8,           intake,        tmotorVex393_MC29, openLo
#pragma config(Motor,  port9,           claw,          tmotorVex393_MC29, openLo
#pragma config(Motor,  port10,          rotator,       tmotorVex393_HBridge, ope
//*!!Code automatically generated by 'ROBOTC' configuration wizard

#pragma competitionControl(Competition)

//This file contains the basic structure of drive control and autonomous.


//inputs for auton functions
const int REDSIDE = 1;
const int BLUESIDE = -1;


//lift height to avoid being shot by shooter
const int liftOutOfTheWayHeight = 800;

const int fullPower = 127;

//minimum joystick threshold to run drive
const int driveThreshold = 20;

const int liftLowPos = 850;

const int clawOpenPos = 550;
const int clawClosePos = -100;

#include "Vex_Competition_Includes.c"
#include "functions.c"
#include "auton.c"

//LCD code variables
int autonIndex = 6;
string mainBattery;
const int numAutons = 8;
string autons[numAutons] = {
    "none",
    "prog",
    "redNearPark",
    "blueNearPark",
    "redNearCap",
    "blueNearCap",
```

```
    "redFar",
    "blueFar"
};
const short leftButton = 1;
const short rightButton = 4;

void pre_auton() {
    datalogClear();
    displayNextLCDString("Gyro Calibrating");
    calibrateGyro();
    bStopTasksBetweenModes = true;
    bLCDBacklight = true;
    while (bIfiRobotDisabled) {
        clearLCDLine(0); // Clear line 1 (0) of the LCD
        clearLCDLine(1); // Clear line 2 (1) of the LCD
        displayLCDString(0, 0, "Primary: ");
        sprintf(mainBattery, "%1.2f%c", nImmediateBatteryLevel / 1000.0, 'V');
        displayNextLCDString(mainBattery);
        displayLCDString(1, 0, autons[autonIndex]);

        //adjust selected auton
        if (nLCDButtons == leftButton) {
            autonIndex--;
            waitForRelease();
        } else if (nLCDButtons == rightButton) {
            autonIndex++;
            waitForRelease();
        } else if (nLCDButtons == 2) {
            clearLCDLine(1);

            displayNextLCDString("Gyro Calibrating");
            wait1Msec(1000);
            calibrateGyro(); //calibrates the gyroscopic sensor
        }

        //loop auton over
        autonIndex = autonIndex % numAutons;
        if (autonIndex == -1) autonIndex = numAutons - 1;
        wait1Msec(25);
    }
    displayLCDCenteredString(1, "Calibrating...")

    drivePID.pGain = 0.25;
    drivePID.iGain = 0;
    drivePID.dGain = 0;

    /**liftPID.pGain=0.15;
    liftPID.iGain=0.0002;
    liftPID.dGain=0.01;
    liftPID.iMin=-20000;
    liftPID.iMax=20000;
    liftPID.constant=4;**/
    liftPID.pGain = 0.15;
    liftPID.iGain = 0.0;
    liftPID.dGain = 0.1;
    liftPID.iMin = -200000;
    liftPID.iMax = 200000;
    liftPID.constant = 4;
    liftPID.timeDuration = 10;
```

```
    gyroPID.pGain = 0.25;
    gyroPID.iGain = 0.0;
    gyroPID.dGain = 0;

    clawPID.pGain = 0.1;
    clawPID.iGain = 0;
    clawPID.dGain = 0;

    rotatorPID.pGain = 0.2;
    rotatorPID.iGain = 0;
    rotatorPID.dGain = 0;
}

task autonomous() {
    bLCDBacklight = false;
    SensorValue[gyro] = 0;
    if (autonIndex == 1) prog();
    else if (autonIndex == 2) nearAutonPark(REDSIDE);
    else if (autonIndex == 3) nearAutonPark(BLUESIDE);
    else if (autonIndex == 4) nearAutonCap(REDSIDE);
    else if (autonIndex == 5) nearAutonCap(BLUESIDE);
    else if (autonIndex == 6) farAuton(REDSIDE);
    else if (autonIndex == 7) farAuton(BLUESIDE);

    //prog();
    //farAuton(REDSIDE);
}

//Moves the lift to a height to not block the lifte
void getLiftOutOfTheWay() {
    if (liftPID.target <= liftOutOfTheWayHeight) liftPID.target = liftOutOfTheWa
}

task usercontrol() {
    // Start subsystem tasks
    //farAuton(REDSIDE);
    //while(1){}

    startTask(liftControl);
    startTask(rotatorTask);
    startTask(clawTask);
    //startTask(driveLocker);
    rotatorPID.target = rotatorLowPos;
    clawPID.target = clawClosePos;
    while (true) {
        displayLCDString(0, 0, "Primary: ");
        sprintf(mainBattery, "%1.2f%c", nImmediateBatteryLevel / 1000.0, 'V');
        displayNextLCDString(mainBattery);

        if (vexRT[btn7d]) {
            //stopAllTasks()
            //farAuton(REDSIDE)
            //pTurn(900);
        }

        // Drive control
        if (abs(vexRT[Ch1]) > driveThreshold || abs(vexRT[Ch3]) > driveThresholc
            lockDrive = false;
```

```
            runLeftDrive(vexRT[Ch3]);
            runRightDrive(vexRT[Ch2]);
        } else {
            lockDrive = true;
            drive(0);
        }

        // Shooter control
        if (vexRT[Btn6u] || vexRT[Btn8DXmtr2]) {
            motor[slingshot] = fullPower;
            getLiftOutOfTheWay()
        }
        /**else if(SensorValue[shooterLimit]==1){
            motor[slingshot]=50;
              }**/
        else {
            motor[slingshot] = 0;
        }

        // Intake control
        if (vexRT[Btn5UXmtr2] || vexRT[btn5u]) {
            motor[intake] = fullPower;
            getLiftOutOfTheWay();
        } else if (vexRT[Btn5d] || vexRT[Btn5DXmtr2]) {
            motor[intake] = -fullPower;
            getLiftOutOfTheWay();
        } else {
            motor[intake] = 0;
        }

        // Rotator control
        if (vexRT[Btn8LXmtr2] || vexRT[btn8l]) {
            liftPID.target += 500;
            wait1Msec(500);
            if (rotatorPID.target == rotatorLowPos) rotatorPID.target = rotatorH
            else if (rotatorPID.target == rotatorHighPos) rotatorPID.target = ro
        }
        if (vexRT[Btn8RXmtr2] || vexRT[btn8r]) {
            clawPID.target = clawClosePos;
            wait1Msec(200);
            liftPID.target += 300;
            if (rotatorPID.target == rotatorLowPos) rotatorPID.target = rotatorH
            else if (rotatorPID.target == rotatorHighPos) rotatorPID.target = ro
            wait1Msec(300);
            liftPID.target -= 300;
            wait1Msec(200);
            clawPID.target = clawOpenPos;
        }

        // Claw control
        if (vexRT[Btn7RXmtr2] || vexRT[btn7r]) {
            clawPID.target = clawOpenPos - 200;
            clawIdle = false;
        } else if (vexRT[Btn7LXmtr2] || vexRT[btn7l]) {
            //open the claw less when the rotator is in the low position
            if (rotatorPID.target < 200) clawPID.target = clawClosePos - 200;
            else clawPID.target = clawClosePos;

            clawIdle = false;
```

```
        }
        if (vexRT[Btn8UXmtr2]) rotatorPID.target = rotatorHighPos;
        else if (vexRT[Btn8DXmtr2]) rotatorPID.target = rotatorLowPos;

        if (vexRT[Btn7DXmtr2]) clawIdle = true;

        if (vexRT[btn8d]) motor[port1] = -127;
        else motor[port1] = 0;
    }

}
```