#pragma config(Sensor, in1, IntakeAngle, sensorPotentiometer)

#pragma config(Sensor, in2, CBAngle, sensorPotentiometer)

#pragma config(Sensor, dgtl1, RightEncoder, sensorQuadEncoder)

#pragma config(Sensor, dgtl3, LeftEncoder, sensorQuadEncoder)

#pragma config(Sensor, I2C\_1, RightEncoder, sensorNone)

#pragma config(Sensor, I2C\_2, LeftEncoder, sensorNone)

#pragma config(Motor, port1, FrontRight, tmotorVex393\_HBridge, openLoop, reversed)

#pragma config(Motor, port2, FrontRight, tmotorVex393\_MC29, openLoop, reversed)

#pragma config(Motor, port3, BackRight, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port4, FrontLeft, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port5, FrontLeft, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port6, BackLeft, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port7, FourBar, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port8, Chainbar, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port9, Claw, tmotorVex393\_MC29, openLoop)

#pragma config(Motor, port10, Intake, tmotorVex393\_HBridge, openLoop)

//\*!!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 method as "competition"

#pragma competitionControl(Competition)

//Main competition background code...do not modify!

#include "Vex\_Competition\_Includes.c"

/\*---------------------------------------------------------------------------\*/

/\* 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 pre\_auton()

{

// 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. \*/

/\* \*/

/\* You must modify the code to add your own robot specific commands here. \*/

/\*---------------------------------------------------------------------------\*/

task autonomous()

{

// ..........................................................................

wait1Msec(2000);

//Clear the encoders associated with the left and right motors

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

SensorValue[IntakeAngle] = 0;

SensorValue[CBAngle] = 0;

motor[Claw] = 80;

motor[Chainbar] = 100;

wait1Msec(800);

motor[Chainbar] = 0;

while(SensorValue[LeftEncoder] > -675)

{

motor[FrontLeft] = motor[BackLeft] = 75;

motor[FrontRight] = motor[BackRight] = 75;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

motor[Intake] = -80;

wait1Msec(1200);

motor[Intake] = 0;

while(SensorValue[LeftEncoder] > -800)

{

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 80;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

motor[Intake] = 120;

wait1Msec(1200);

motor[Intake] = 0;

while(SensorValue[LeftEncoder] < 600)

{

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = -80;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

while(SensorValue[RightEncoder] > -400)

{

motor[FrontRight] = motor[BackRight] = 80;

motor[FrontLeft] = motor[BackLeft] = -80;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

while(SensorValue[LeftEncoder] > -500)

{

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 80;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

while(SensorValue[RightEncoder] > -240)

{

motor[FrontRight] = motor[BackRight] = 80;

motor[FrontLeft] = motor[BackLeft] = -80;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

while(SensorValue[LeftEncoder] > -700)

{

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 100;

}

SensorValue[RightEncoder] = 0;

SensorValue[LeftEncoder] = 0;

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

motor[Chainbar] = -70;

wait1Msec(400);

motor[Chainbar] = 0;

motor[Claw] = 0;

motor[Claw] = -60;

wait1Msec(700);

motor[Chainbar] = 100;

wait1Msec(300);

motor[Claw] = 100;

wait1Msec(800);

motor[Chainbar] = 0;

motor[Intake] = -80;

wait1Msec(600);

while(SensorValue[LeftEncoder] < 1000)

{

motor[FrontLeft] = motor[BackLeft] = -75;

motor[FrontRight] = motor[BackRight] = -75;

}

/\*

while(SensorValue[LeftEncoder] < 2000)

{

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = -127;

}

motor[FrontLeft] = motor[BackLeft] = motor[FrontRight] = motor[BackRight] = 0;

}

\*/

}

task usercontrol()

{

while (true)

{

if(vexRT[Btn7U] == 1)

{

motor[FourBar] = 100;

}

if(vexRT[Btn7D] == 1)

{

motor[FourBar] = -100;

}

if(vexRT[Btn7U] == 0&&vexRT[Btn7D] == 0)

{

motor[FourBar] = 0;

}

motor[FrontLeft] = motor[BackLeft] = vexRT[Ch3] + vexRT[Ch1];

motor[FrontRight] = motor[BackRight] = vexRT[Ch3] - vexRT[Ch1];

if(vexRT[Btn8R] == 1)

{

motor[Claw] = 80;

}

if(vexRT[Btn8D] == 1)

{

motor[Claw] = -80;

}

if(vexRT[Btn8R] == 0 && vexRT[Btn8D] == 0)

{

motor[Claw] = 0;

}

if(vexRT[Btn6D] == 1)

{

motor[Chainbar] = 100;

}

if(vexRT[Btn5D] == 1)

{

motor[Chainbar] = -100;

}

if(vexRT[Btn5D] == 0 && vexRT[Btn6D] == 0)

{

motor[Chainbar] = 0;

}

if(vexRT[Btn6U] == 1 && SensorValue(IntakeAngle)>1650)

{

motor[Intake] = -100;

}

if(vexRT[Btn5U] == 1)

{

motor[Intake] = 100;

}

if(vexRT[Btn5U] == 0 && vexRT[Btn6U] == 0)

{

motor[Intake] = 0;

}

}

}