```
#pragma config(Sensor, dgtl1,    leftEncoder,       sensorQuadEncoder)
#pragma config(Sensor, dgtl3,    rightEncoder,      sensorQuadEncoder)
#pragma config(Motor,    port1,           rightMotorF,      tmotorVex393_HBridge, openLoop)
#pragma config(Motor,    port2,           rightMotorB,      tmotorVex393_MC29, openLoop)
#pragma config(Motor,    port3,           rightTowers,      tmotorVex393_MC29, openLoop,
reversed)
#pragma config(Motor,    port4,           leftTowers,       tmotorVex393_MC29, openLoop)
#pragma config(Motor,    port5,           mobileGoal,       tmotorVex393_MC29, openLoop,
driveRight)
#pragma config(Motor,    port6,           flipperLeft,      tmotorVex393_MC29, openLoop, driveLeft)
#pragma config(Motor,    port7,           flipperRight,     tmotorVex393_MC29, openLoop,
driveRight)
#pragma config(Motor,    port8,           clawMotor,        tmotorVex393_MC29, openLoop,
reversed)
#pragma config(Motor,    port9,           leftMotorF,       tmotorVex393_MC29, openLoop,
reversed, driveLeft)
#pragma config(Motor,    port10,          leftMotorB,       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 ()

{

}

void driveFront(int front)

{

	int masterPower = 120;	//do not set this at 127 because the slave power will not be able to
go over

	int slavePower = 120;

	int masterPower1= 120;

	int slavePower1 = 120;
```

```
int error1 = 0;

int kp1 = 5;


//Reset the encoders.

SensorValue[leftEncoder] = 0;

SensorValue[rightEncoder] = 0;


while((SensorValue[leftEncoder]) < front)// move forward to stars

{

        //Set the motor powers to their respective variables.

        motor[leftMotorF] = masterPower;

        motor[rightMotorF] = slavePower;

        motor[leftMotorB] = masterPower1;

        motor[rightMotorB] = slavePower1;




        error1 = SensorValue[leftEncoder] - SensorValue[rightEncoder];

        slavePower1 += error1 / kp1;

        wait1Msec(100);

}

motor[leftMotorF] = -20;

motor[rightMotorF] = -20;

motor[leftMotorB] = -20;

motor[rightMotorB] = -20;

wait1Msec(25);
```

```
        motor[leftMotorF] = 0;

        motor[rightMotorF] = 0;

        motor[leftMotorB] = 0;

        motor[rightMotorB] = 0;

}


void driveBack(int back)

{

        //The powers we give to both motors. masterPower will remain constant while slavePower will
change so that

        //the right wheel keeps the same speed as the left wheel.

        int masterPower = -125;

        int slavePower = -125;

        int masterPower1 = -125;

        int slavePower1 = -125;


        //Essentially the difference between the master encoder and the slave encoder. Negative if
slave has

        //to slow down, positive if it has to speed up. If the motors moved at exactly the same speed,
this

        //value would be 0.

        int error = 0;


        //'Constant of proportionality' which the error is divided by. Usually this is a number between 1
and 0 the

        //error is multiplied by, but we cannot use floating point numbers. Basically, it lets us choose
how much
```

```
//the difference in encoder values effects the final power change to the motor.
int kp = 15;

//Reset the encodback
SensorValue[leftEncoder] = 0;
SensorValue[rightEncoder] = 0;
wait1Msec(500);
//Repeat ten times a second.
while((SensorValue[leftEncoder]*-1) < back)
{
	//Set the motor powers to their respective variables.
	motor[leftMotorF] = masterPower;
	motor[rightMotorF] = slavePower;
	motor[leftMotorB] = masterPower1;
	motor[rightMotorB] = slavePower1;

	//This is where the magic happens. The error value is set as a scaled value representing the amount the slave
	//motor power needs to change. For example, if the left motor is moving faster than the right, then this will come
	//out as a positive number, meaning the right motor has to speed up.
	error = SensorValue[leftEncoder] - SensorValue[rightEncoder];

	//This adds the error to slavePower, divided by kp. The '+=' operator literally means that this expression really says
	//"slavePower = slavepower + error / kp", effectively adding on the value after the operator.
```

```
            //Dividing by kp means that the error is scaled accordingly so that the motor value does
not change too much or too

            //little. You should 'tune' kp to get the best value. For us, this turned out to be around 5.

            slavePower += error / kp;




            //Makes the loop repeat ten times a second. If it repeats too much we lose accuracy due
to the fact that we don't have

            //access to floating point math, however if it repeats to little the proportional algorithm
will not be as effective.

            //Keep in mind that if this value is changed, kp must change accordingly.

            wait1Msec(100);

        }

        motor[leftMotorF] = 20;

        motor[rightMotorF] = 20;

        motor[leftMotorB] = 20;

        motor[rightMotorB] = 20;

        wait1Msec(250);

        motor[leftMotorF] = 0;

        motor[rightMotorF] = 0;

        motor[leftMotorB] = 0;

        motor[rightMotorB] = 0;

}

void pointTurnLeft(int turnDistanceLeft)

{
```

```
        SensorValue(rightEncoder) = 0; // Reset the right motor encoder

        SensorValue(leftEncoder) = 0;    // Reset the left motor encoder

        wait1Msec(100); //wait one second


        while(SensorValue[rightEncoder] < turnDistanceLeft)    // While the Motor Encoder of leftMotor
has not yet reached 360 counts:

        {

                motor[rightMotorF] = 55;      // right motor is given a power level of 75.

                motor[leftMotorF] = -55;

                motor[rightMotorB] = 55;

                motor[leftMotorB] = -55;

        }



        motor[rightMotorF] = 0;        // right motor is given a power level of 0 (stop).

        motor[leftMotorF] = 0; // left motor is given a power level of 0 (stop).

        motor[rightMotorB] = 0;

        motor[leftMotorB] = 0;

        wait1Msec(100);      //pause one second


}

void pointTurnRight(int turnDistanceRight)

{


        SensorValue(rightEncoder) = 0; // Reset the right motor encoder

        SensorValue(leftEncoder) = 0;    // Reset the left motor encoder
```

```
	wait1Msec(100); //wait one second


	while(SensorValue[leftEncoder] < turnDistanceRight)    // While the Motor Encoder of leftMotor
has not yet reached 360 counts:

	{

		motor[rightMotorF] = -55;      // right motor is given a power level of 75.

		motor[leftMotorF] = 55;

		motor[rightMotorB] = -55;

		motor[leftMotorB] = 55;

	}



	motor[rightMotorF] = 0;      // right motor is given a power level of 75.

	motor[leftMotorF] = 0;

	motor[rightMotorB] = 0;

	motor[leftMotorB] = 0;      // left motor is given a power level of 0 (stop).

	wait1Msec(100);      //pause one second

}

void mobileliftup ()

{

	motor[mobileGoal] = 85;

	wait1Msec(1275);

	motor[mobileGoal] = 0;

}

void mobileliftdown ()

{
```

```
	{
		motor[mobileGoal] = -127;
		wait1Msec(700);
		motor[mobileGoal] = 0;
	}

	task mobileliftdown2 ()
	{
		motor[mobileGoal] = -127;
		wait1Msec(700);
		motor[mobileGoal] = 0;
	}
	task thomas ()
	{
		while (1==1)
		{
			motor[rightMotorF] = vexRT[Ch3];
			motor[rightMotorB] = vexRT[Ch3];
			motor[leftMotorF] = vexRT[Ch2];
			motor[leftMotorB] = vexRT[Ch2];
		}
	}
	task lift1 ()
	{
		while(1==1)
		{
```

```
            if (vexRT[Btn5UXmtr2] == 1)             //lift up
            {
                while (vexRT[Btn5UXmtr2] == 1)
                {
                    motor[rightTowers] = 90;
                    motor[leftTowers] = -90;
                }
                motor[rightTowers] = 5;
                motor[leftTowers] = 5;
            }
        }
}
task lift2 ()
{
        while(1==1)
        {
            if (vexRT[Btn5DXmtr2] == 1)             //lift down
            {
                while (vexRT[Btn5DXmtr2] == 1)
                {
                    motor[rightTowers] = -90;
                    motor[leftTowers] = 90;
                }
                motor[rightTowers] = 5;
                motor[leftTowers] = 5;
```

```
		}
	}
}
task flipper1 ()
{
	while(1==1)
	{
		if(vexRT[Btn6UXmtr2] == 1)
		{
			while (vexRT[Btn6UXmtr2] == 1)		//Flipper goes outward towards cones
on field or preloads
			{
				motor[flipperLeft] = -45;
				motor[flipperRight] = 90;
			}
			motor[flipperLeft] = 10;
			motor[flipperRight] = 10;
		}
	}
}
task flipper2 ()
{
	while(1==1)
	{
		if(vexRT[Btn6DXmtr2] == 1)
		{
```

```
                while (vexRT[Btn6DXmtr2] == 1)          //Flipper goes intward towards cones
on field or preloads
                {
                        motor[flipperLeft] = 45;
                        motor[flipperRight] = -90;
                }
                motor[flipperLeft] = 10;
                motor[flipperRight] = 10;
        }
    }
}
/*---------------------------------------------------------------------------*/
/*                                                                           */
/*                              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()
{

    startTask(mobileliftdown2);                         //puts mobile lift down
to get base
```

```
driveFront(1350);

mobileliftup();
//picks up mobile base

driveBack(1300);

pointTurnRight(415);

driveFront(500);

pointTurnRight(295);

motor[rightMotorF] = 127;                                    //rushing to the
20pt zone

motor[leftMotorF] = 127;

motor[leftMotorB] = 127;

motor[rightMotorB] = 127;

wait1Msec(1300);

motor[rightMotorF] = 0;

motor[leftMotorF] = 0;

motor[leftMotorB] = 0;

motor[rightMotorB] = 0;

wait1Msec(10);

motor[mobileGoal] = -127;                                    //scoring in the
20pt zone

driveBack(50);

driveFront(50);

driveBack(400);

motor[mobileGoal] = 0;

wait1Msec(10);

pointTurnLeft(400);
//turns to the 2nd mobile base
```

```
	driveFront(500);

	pointTurnLeft(275);

	driveFront(500);

	mobileliftup();
	//picks up base2

	pointTurnLeft(550);

	driveFront(1400);

	mobileliftdown();
	//scores base2 in the 10pt zone

	/*driveBack(1300);

	pointTurnRight(200);

	driveFront(1000);

	mobileliftup();
	//picks up base3

	driveFront(100);

	pointTurnLeft(350);
	//turns torward 10pt zone

	driveFront(1150);

	mobileliftdown();

	driveBack(500);

*/
```

```
        // Remove this function call once you have "real" code.

        AutonomousCodePlaceholderForTesting();

}


/*---------------------------------------------------------------------------*/

/*                                                                           */

/*                              User Control Task                            */

/*                                                                           */

/*  This task is used to control your robot during the user control phase of */

/*  a VEX Competition.                                                       */

/*                                                                           */

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

/*---------------------------------------------------------------------------*/


task usercontrol()

{

        startTask(thomas);


        startTask(lift1);


        startTask(lift2);


        startTask(flipper1);
```

```
	}

startTask(flipper2);

while (true)
{
	//driver control
	if (vexRT[Btn5UXmtr2] == 1)			//lift up
	{
		while (vexRT[Btn5UXmtr2] == 1)
		{
			motor[rightTowers] = 90;
			motor[leftTowers] = -90;
		}
		motor[rightTowers] = 5;
		motor[leftTowers] = 5;
	}
	if (vexRT[Btn5DXmtr2] == 1)			//lift down
	{
		while (vexRT[Btn5DXmtr2] == 1)
		{
			motor[rightTowers] = -90;
			motor[leftTowers] = 90;
		}
		motor[rightTowers] = 5;
		motor[leftTowers] = 5;
	}
```

```
		if(vexRT[Btn6UXmtr2] == 1)
		{
			while (vexRT[Btn6UXmtr2] == 1)		//Flipper goes outward towards cones
on field or preloads
			{
				motor[flipperLeft] = -45;
				motor[flipperRight] = 90;
			}
			motor[flipperLeft] = 10;
			motor[flipperRight] = 10;
		}
		if(vexRT[Btn6DXmtr2] == 1)
		{
			while (vexRT[Btn6DXmtr2] == 1)		//Flipper goes intward towards cones
on field or preloads
			{
				motor[flipperLeft] = 45;
				motor[flipperRight] = -90;
			}
			motor[flipperLeft] = 10;
			motor[flipperRight] = 10;
		}
		if(vexRT[Btn8UXmtr2] == 1)
		{
			while (vexRT[Btn8UXmtr2] == 1)
			{
```

```
                    motor[clawMotor] = 10;                  // This allows the claw to suck
in
                }
                motor[clawMotor] = 1;
            }
            if(vexRT[Btn8DXmtr2] == 1)
            {
                while (vexRT[Btn8DXmtr2] == 1)
                {
                    motor[clawMotor] = -127;                                // This
allows the claw to suck out
                }
                motor[clawMotor] = 0;
            }
            if(vexRT[Btn6U] == 1)
            {
                while (vexRT[Btn6U] == 1)
                {
                    motor[mobileGoal] = 90;                         // This allows
the Mobile Goal Lift to go Up
                }
                motor[mobileGoal] = 5;
            }
            if(vexRT[Btn6D] == 1)
            {
                while (vexRT[Btn6D] == 1)
                {
```

```
			motor[mobileGoal] = -90;				// This allows the Mobile Goal Lift to go Down
		}
		motor[mobileGoal] = 5;
	}
}
}
```