```
#ifndef MAIN_H_
#define MAIN_H_

#include <API.h>

#ifdef __cplusplus
extern "C" {
    #endif

    #define PI 3.14159265359
    #define OFF 0

    //Drive Motors
    #define backRight 3
    #define frontLeft 6
    #define backLeft 7
    #define frontRight 8

    //Lift Motors
    #define lowerRightLift 2
    #define upperLift 4
    #define lowerLeftLift 9

    //Claw Motors
    #define fingerY 5

    //Digital Sensors
    #define liftQuadPort 1
    #define leftFingerSwitchPort 3
    #define rightFingerSwitchPort 4
    #define rightQuadPort 5
    #define leftQuadPort 7

    //Analog Sensors
    #define gyroOnePort 5
    #define gyroTwoPort 4
    #define liftPot 6
    #define potOne 7
    #define potTwo 8

    #define MOTOR_NUM 10
    #define MOTOR_MAX_VALUE 127
    #define MOTOR_MIN_VALUE -127
    #define MOTOR_DEFAULT_SLEW_RATE 20
    #define MOTOR_FAST_SLEW_RATE 256
    #define MOTOR_TASK_DELAY 20
    #define MOTOR_DEADBAND 10

    #define ANALOG_DEADZONE 10

    #define MID_HEIGHT 600
    #define HIGH_HEIGHT 650
    #define DOWN_HEIGHT 20

    //bool initialized;

    int motorSlew[MOTOR_NUM]; //Array containing the slew rates for each individual motor port
    int motorReq[MOTOR_NUM]; //Array containing the requested speed for each indivual motor port (-127 to 127)

    //Enumeration defining autonomous movement direction
    enum WheelDirection{
        FORWARD,
        BACKWARD,
        LEFT,
        RIGHT,
    };
```

```
//Theoretical Encoder Clicks for turning (Not accurate in practice due to wheels slipping)
float WHEEL_CIR;
float TOLERANCE;
int FULL;
int QUARTER;
int HALF;
int THREE_QUARTER;

//WheelMonitorTask variables
int wheelTargetTicks;
enum WheelDirection wheelDir;
bool runWheels;
int DRIVEBASE_POWER;
float TURN_MULTIPLIER;

//LiftMonitorTask variables
bool runLift;
int liftTargetTicks;
int LIFT_POWER;

bool liftPIDRunning;
float liftPGain;
float liftIGain;
float liftDGain;
float liftDerivative;
float lastLiftError;
int liftError;
int liftLastError;
int liftCumError;
int liftOutput;
float liftDeltaTime;


//ClawMonitorTask variables
bool downPressure;
bool runFinger;
bool fingerNeedsToOpen;
bool clawClosing;
int CLAW_POWER;

//Index of the autonomous routine to run based on the two potentiometers mounted on the back of the robot
int autonSelection;

bool useGyro;

//Quadrature Encoders
Encoder liftQuad;
Encoder rightQuad;
Encoder leftQuad;
Gyro gyroOne;
Gyro gyroTwo;

TaskHandle clawMonitorHandle;
TaskHandle wheelMonitorHandle;
TaskHandle liftMonitorHandle;
TaskHandle liftPIDHandle;
TaskHandle motorSlewHandle;
TaskHandle taskMonitorHandle;

Mutex motorReqMutex;
Mutex motorMutexes[10];

Mutex runWheelsMutex;
Mutex wheelDirMutex;
Mutex driveTicksMutex;
```

```
    Mutex runLiftMutex;
    Mutex liftTicksMutex;

    Mutex runFingerMutex;
    Mutex downPressureMutex;
    Mutex clawClosingMutex;

    Mutex useGyroMutex;

    int programSelected(int segments);
    int clamp(int var, int min, int max);

    void motorSlewTask(void *parameter);
    void waitForTasks();
    void stopAllMotors();

    void wheelMonitorTask(void *parameter);
    void setSyncMove(enum WheelDirection d,int targetTicks, bool enableGyro);
    void dLeft(bool backwards, bool bypassSlew);
    void dRight(bool backwards, bool bypassSlew);
    void strafeRight(int millis);
    void strafeLeft(int millis);
    void analogDrive();
    void stopLeft();
    void stopRight();
    void stopDrive();

    void liftMonitorTask(void *parameter);
    void liftPID(void *parameter);
    void setSyncLift(int targetTicks);
    void dLift(bool down);
    void stopLift();

    void clawMonitorTask(void *parameter);
    void closeClaw(int millis);
    void openClaw();

    void zeroDriveSensors();
    void zeroAllSensors();

    void autonZero();
    void autonOne();
    void autonTwo();
    void autonThree();
    void autonFour();
    void autonFive();
    void autonSix();
    void autonSeven();
    void autonEight();
    void autonNine();
    void autonTen();
    void autonEleven();
    void autonTwelve();
    void autonThirteen();
    void autonFourteen();

    void autonomous();
    void initializeIO();
    void initialize();
    void operatorControl();

    #ifdef __cplusplus
}
#endif
```

```
#endif
```