```c
/**
 * \file main.h
 *
 * Contains common definitions and header files used throughout your PROS
 * project.
 *
 * Copyright (c) 2017-2021, Purdue University ACM SIGBots.
 * All rights reserved.
 *
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */

#ifndef _PROS_MAIN_H_
#define _PROS_MAIN_H_

/**
 * If defined, some commonly used enums will have preprocessor macros which give
 * a shorter, more convenient naming pattern. If this isn't desired, simply
 * comment the following line out.
 *
 * For instance, E_CONTROLLER_MASTER has a shorter name: CONTROLLER_MASTER.
 * E_CONTROLLER_MASTER is pedantically correct within the PROS styleguide, but
 * not convienent for most student programmers.
 */
#define PROS_USE_SIMPLE_NAMES

/**
 * If defined, C++ literals will be available for use. All literals are in the
 * pros::literals namespace.
 *
 * For instance, you can do `4_mtr = 50` to set motor 4's target velocity to 50
 */
#define PROS_USE_LITERALS

#include "api.h"

/**
 * You should add more #includes here
 */
#include "okapi/api.hpp"

//#include "pros/api_legacy.h"

/**
 * If you find doing pros::Motor() to be tedious and you'd prefer just to do
 * Motor, you can use the namespace with the following commented out line.
 *
 * IMPORTANT: Only the okapi or pros namespace may be used, not both
 * concurrently! The okapi namespace will export all symbols inside the pros
 * namespace.
 */
// using namespace pros;
// using namespace pros::literals;
using namespace okapi;

```

```

/**
 * Prototypes for the competition control tasks are redefined here to ensure
 * that they can be called from user code (i.e. calling autonomous from a
 * button press in opcontrol() for testing purposes).
 */
#ifdef __cplusplus
extern "C" {
#endif
void autonomous(void);
void initialize(void);
void disabled(void);
void competition_initialize(void);
void opcontrol(void);
#ifdef __cplusplus
}
#endif


#ifdef __cplusplus
/**
 * You can add C++-only headers here
 */

#include <iostream>
#include <unordered_map>

#endif

#endif  // _PROS_MAIN_H_
```

```c
/**
 * \file api.h
 *
 * PROS API header provides high-level user functionality
 *
 * Contains declarations for use by typical VEX programmers using PROS.
 *
 * This file should not be modified by users, since it gets replaced whenever
 * a kernel upgrade occurs.
 *
 * Copyright (c) 2017-2021, Purdue University ACM SIGBots.
 * All rights reserved.
 *
 * This Source Code Form is subject to the terms of the Mozilla Public
 * License, v. 2.0. If a copy of the MPL was not distributed with this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/.
 */

#ifndef _PROS_API_H_
#define _PROS_API_H_

#ifdef __cplusplus
#include <cerrno>
#include <cmath>
#include <cstdbool>
#include <cstddef>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#else /* (not) __cplusplus */
#include <errno.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#endif /* __cplusplus */

#define PROS_VERSION_MAJOR 3
#define PROS_VERSION_MINOR 5
#define PROS_VERSION_PATCH 4
#define PROS_VERSION_STRING "3.5.4"

#define PROS_ERR (INT32_MAX)
#define PROS_ERR_F (INFINITY)

#include "pros/adi.h"
#include "pros/colors.h"
#include "pros/distance.h"
#include "pros/ext_adi.h"
#include "pros/gps.h"
#include "pros/imu.h"
#include "pros/llemu.h"
#include "pros/misc.h"
```

```c
#include "pros/motors.h"
#include "pros/optical.h"
#include "pros/rtos.h"
#include "pros/rotation.h"
#include "pros/screen.h"
#include "pros/vision.h"

#ifdef __cplusplus
#include "pros/adi.hpp"
#include "pros/distance.hpp"
#include "pros/gps.hpp"
#include "pros/imu.hpp"
#include "pros/llemu.hpp"
#include "pros/misc.hpp"
#include "pros/motors.hpp"
#include "pros/optical.hpp"
#include "pros/rotation.hpp"
#include "pros/rtos.hpp"
#include "pros/screen.hpp"
#include "pros/vision.hpp"
#endif

#endif  // _PROS_API_H_
```

```cpp
#ifndef _BUTTON_
#define _BUTTON_
#include "main.h"


//structure to store button state and count
struct But {
  bool state;
  int count;
};

//class to handle all controller buttons
class Button {
private:
  std::unordered_map<okapi::ControllerDigital, But> buttons;
public:
  Button();
  okapi::ControllerDigital buttonList[9] =
{okapi::ControllerDigital::L1,okapi::ControllerDigital::A, okapi::ControllerDigital::X,
okapi::ControllerDigital::right, okapi::ControllerDigital::R1, okapi::ControllerDigital::L2,
okapi::ControllerDigital::R2, okapi::ControllerDigital::B, okapi::ControllerDigital::left};
  void handleButtons(okapi::Controller controller);
  int getCount(okapi::ControllerDigital id);
  bool getPressed(okapi::ControllerDigital id);
  void init();
};

#endif
```

```cpp
#ifndef _DRIVE_
#define _DRIVE_

#define WHEELDIM 4_in
#define WHEELTRACK 10_in

#include "main.h"
#include "ports.h"

//chassis controller wrapper with drive utilities
class Drive {
private:
  std::shared_ptr<okapi::OdomChassisController> chassis;
  // okapi::IntegratedEncoder enc;
  int speedfactor;
public:
  Drive();
  double getX();
  double getY();
  double getHeading();
  void run(double forward, double strafe, double turn);
  okapi::OdomState getState();
  void runWithController();
  void runTankArcade(double forward, double turn);
  void runTank(double left, double right);
  void reverseOrientation(int ori);
  void setMode(okapi::AbstractMotor::brakeMode brakeMode);
  // double getEncoder();
};

#endif
```

```cpp
#ifndef _EFFECTORS_
#define _EFFECTORS_

#include "main.h"

//class for two-bar actuation
class Effectors {
private:
  //two bar port
  okapi::Motor motors[1] = {okapi::Motor(-20)};
  int encPositions[3][3];
  int prevCounts[3];
  bool goalFinal = false;
  bool spikeUp = false;
public:
  Effectors();
  void run(bool left, bool right, double speed);
  void step(int buttons[3], double speeds[3]);
  void addPosition();
  void runOne(int lift, int pos);
  void runOneToPosition(int lift, int pos);
};

#endif
```

```cpp
#ifndef _INCLUDES_
#define _INCLUDES_

#include "main.h"
#include "pid.h"
#include "drive.h"
#include "pneumatics.h"
#include "button.h"
#include "intake.h"
#include "effectors.h"
#include "PurePursuitPathGen.h"
#include "PurePursuitFollower.h"

//extern definition of global objects

extern Drive *drive;
extern Pneumatics *pneum;
extern Effectors effectors;
extern Intake *intake;
extern Button *buttons;

#endif
```

```cpp
#ifndef _INTAKE_
#define _INTAKE_

#include "main.h"


//Class for intake and four bar actuation
class Intake {
private:
  okapi::Motor m;
  bool dir;
  bool moving;
  std::vector<int> encPositions;
  int prevCount = 0;
  int upper;
  int lower;
  bool limits = false;
public:
  Intake(double port);
  void run(bool left, bool right, double speed);
  void moveTarget(double enc);
  void setTarget(double enc);
  void stepAbsolute(int count, double speed);
  void addPosition(int pos);
  void step();
  void setLimits(int upper, int lower);
  void handle(int count, double speed);
};



#endif
```

```cpp
#ifndef _ODOMETRY_
#define _ODOMETRY_

#include "main.h"
#include "ports.h"

class IMUOdometry {
public:
  //Odometry(ADIEncoder left, ADIEncoder right, ADIEncoder back, pros::IMU imu, double backdistance, double track);
  OdomState step();
private:
  pros::IMU imu;
  //ADIEncoder left, right, back;
  int prevleft, prevright, prevback;
  OdomState prevState;



};

#endif
```

```cpp
#ifndef _PID_
#define _PID_


//generic constants structure
struct PIDConst {
  double kp, ki, kd;
};

//generic PID class for all PID movements
class PID {
private:
  double kp;
  double ki;
  double kd;
  double totalerr;
public:
  PID(PIDConst constants);
  double step(double err);
};

#endif
```

```cpp
#ifndef PNEUMATICS_H
#define PNEUMATICS_H

#include "main.h"
#include "ports.h"

//class for pneumatics actuation
class Pneumatics {
    private:
        pros::ADIDigitalOut piston;
        bool state = false;
        int prevCount = 0;

    public:
        Pneumatics(uint8_t port);

        // helper methods
        void turnOn();
        void turnOff();

        // opcontrol
        void handle(int count);

        // auton
        void onThenOff(int delay);
        void offThenOn(uint32_t delay);
};

#endif
```

```
#ifndef _PORTS_
#define _PORTS_


//motor ports
#define BOTTOM_RIGHT_MOTOR -11
#define LEFT_MIDDLE_MOTOR 9
#define BOTTOM_LEFT_MOTOR -6
#define TOP_RIGHT_MOTOR -5
#define RIGHT_MIDDLE_MOTOR 12
#define TOP_LEFT_MOTOR -14

//tracking wheel ports
#define LEFT_TRACKING_WHEEL_TOP 'C'
#define LEFT_TRACKING_WHEEL_BOTTOM 'D'
#define RIGHT_TRACKING_WHEEL_TOP 'B'
#define RIGHT_TRACKING_WHEEL_BOTTOM 'A'
#define BACK_TRACKING_WHEEL_TOP 'E'
#define BACK_TRACKING_WHEEL_BOTTOM 'F'


//odom distances between wheels
#define ODOMTRACK 6.9235_in
#define ODOMWHEELDIM 2.81665_in
#define ODOMBACKDISTANCE 6.5_in


//These are lift macros for the index of the lift. Do not touch
#define GOAL_LIFT 0
#define FOUR_BAR 1
#define SPIKE 2
#define INTAKE 3

#define INTAKE_PORT 15
#define FOUR_BAR_FIRST 8
#define IMUPORT 16

#define FRONT_PNEUM 'G'
#define BACK_PNEUM 'F'

#define TRACK 6.875

#endif
```

```cpp
#ifndef _PPFOLLOWER_
#define _PPFOLLOWER_

#include "main.h"
#include "ports.h"
#include "PurePursuitPathGen.h"
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <utility>
#include <array>


//structure for point storage

struct followPoint {
    double x, y, vel;
};


//experimental pure pursuit follower class
class PurePursuitFollower {
public:
    std::vector<followPoint> points;
    okapi::Timer timer = okapi::Timer();
    double lookahead;
    double last_fractional_index = 0;
    int last_closest_point = 0;
    double prev_vel = 0;
    double prev_left = 0;
    double prev_right = 0;
    double prev_time;
    std::pair<double, double> last_lookahead_point;
    std::pair<double, double> lookahead_point;
    followPoint closest_point;
    double curvature;
    double max_accel = 10.0;
    double prevtime;

    void calc_closest_point(double x, double y);
    void calc_lookahead(double x, double y);
    void read_from_file(std::string filename);
    void calc_curvature_at_point(double x, double y, double theta);
    std::array<double, 4> follow_sim(double x, double y, double theta);
    std::array<double, 4> follow(double x, double y, double theta);
    void read(PurePursuitPathGen obj);
    PurePursuitFollower(double lookahead);
};

#endif
```

```cpp
#ifndef _PATH_GEN_
#define _PATH_GEN_

#include "PurePursuitPathGen.h"


#include <vector>
#include <string>
#include <fstream>
#include <iostream>

struct point {
        double x, y, curve, vel, distance;
};

//experimental path generation class
class PurePursuitPathGen {
    public:


    std::vector<point> initial_points;
    std::vector<point> final_points;
    double spacing;
    double a, b, tolerance, max_vel, max_accel;
    int k;

    void interpolate();
    void smooth();
    void calc_distances();
    void calc_curvature();
    void print_path();
    void write_to_file();
    void calc_velocities();
    std::vector<point> get_points();
    PurePursuitPathGen(double spacing, double a, double b, double tolerance, std::vector<point> points, double max_vel, double max_accel, int k);

};

#endif
```