1 Header Files

1.1 include/main.h

```
/**
 * \file main.h
 *
 * Contains common definitions and header files used throughout your PROS
 * project.
 *
 * Copyright (c) 2017-2020, 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>
#endif

#endif  // _PROS_MAIN_H_
```

2 Source Files

2.1 src/main.cpp

```cpp
#include "main.h"

/**
 * A callback function for LLEMU's center button.
 *
 * When this callback is fired, it will toggle line 2 of the LCD text between
 * "I was pressed!" and nothing.
 */
void on_center_button() {
  static bool pressed = false;
  pressed = !pressed;
  if (pressed) {
    pros::lcd::set_text(2, "I was pressed!");
  } else {
    pros::lcd::clear_line(2);
  }
}

/**
 * Runs initialization code. This occurs as soon as the program is started.
 *
 * All other competition modes are blocked by initialize; it is recommended
 * to keep execution time for this mode under a few seconds.
 */
void initialize() {
  pros::lcd::initialize();
  pros::lcd::set_text(1, "Hello PROS User!");

  pros::lcd::register_btn1_cb(on_center_button);
}

/**
 * Runs while the robot is in the disabled state of Field Management System or
 * the VEX Competition Switch, following either autonomous or opcontrol. When
 * the robot is enabled, this task will exit.
 */
void disabled() {}

/**
 * Runs after initialize(), and before autonomous when connected to the Field
 * Management System or the VEX Competition Switch. This is intended for
 * competition-specific initialization routines, such as an autonomous selector
 * on the LCD.
 *
 * This task will exit when the robot is enabled and autonomous or opcontrol
 * starts.
 */
void competition_initialize() {}

/**
```

```
 * Runs the user autonomous code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the autonomous
 * mode. Alternatively, this function may be called in initialize or opcontrol
 * for non-competition testing purposes.
 *
 * If the robot is disabled or communications is lost, the autonomous task
 * will be stopped. Re-enabling the robot will restart the task, not re-start it
 * from where it left off.
 */
void autonomous() {}

/**
 * Runs the operator control code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the operator
 * control mode.
 *
 * If no competition control is connected, this function will run immediately
 * following initialize().
 *
 * If the robot is disabled or communications is lost, the
 * operator control task will be stopped. Re-enabling the robot will restart the
 * task, not resume it from where it left off.
 */
void opcontrol() {
  pros::Controller master(pros::E_CONTROLLER_MASTER);
  pros::Motor left_mtr(1);
  pros::Motor right_mtr(2);

  while (true) {
    pros::lcd::print(0, "%d %d %d", (pros::lcd::read_buttons() & LCD_BTN_LEFT) >> 2,
                     (pros::lcd::read_buttons() & LCD_BTN_CENTER) >> 1,
                     (pros::lcd::read_buttons() & LCD_BTN_RIGHT) >> 0);
    int left = master.get_analog(ANALOG_LEFT_Y);
    int right = master.get_analog(ANALOG_RIGHT_Y);

    left_mtr = left;
    right_mtr = right;
    pros::delay(20);
  }
}
```