```c
// @file main.c
// @brief Main logic for Battery+
//
// Contains the higher level logic code
//
// @author Eric D. Phillips
// @date November 22, 2015
// @bugs No known bugs

#include <pebble.h>
#include "drawing.h"

// Main constants
#define MENU_CELL_COUNT 1
#define MENU_CELL_HEIGHT_TALL 65

// Main data structure
static struct {
  Window      *window;            //< The base window for the application
  MenuLayer   *menu;              //< The main menu layer for the application
} main_data;


// MenuLayer row draw callback
static void prv_menu_row_draw_handler(GContext *ctx, const Layer *layer, MenuIndex *index,
                                      void *context) {
  drawing_render_cell(ctx);   // <-- This gets called and goes to drawing.c
}

// MenuLayer get row count callback
static uint16_t prv_menu_get_row_count_handler(MenuLayer *menu, uint16_t index, void *context) {
  return MENU_CELL_COUNT;
}

// MenuLayer get row height callback
static int16_t prv_menu_get_row_height_handler(MenuLayer *menu, MenuIndex *index, void *context) {
  return MENU_CELL_HEIGHT_TALL;
}


// Initialize menu layer
static void prv_initialize_menu_layer(Layer *window_root, GRect window_bounds) {
  main_data.menu = menu_layer_create(window_bounds);
  menu_layer_set_highlight_colors(main_data.menu, GColorChromeYellow, GColorBlack);
  menu_layer_set_center_focused(main_data.menu, true);
  menu_layer_set_callbacks(main_data.menu, NULL, (MenuLayerCallbacks) {
    .draw_row = prv_menu_row_draw_handler,
    .get_num_rows = prv_menu_get_row_count_handler,
    .get_cell_height = prv_menu_get_row_height_handler,
  });
  menu_layer_set_click_config_onto_window(main_data.menu, main_data.window);
  layer_add_child(window_root, menu_layer_get_layer(main_data.menu));
}

// Initialize the program
static void prv_initialize(void) {
  // initialize window
  main_data.window = window_create();
  Layer *window_root = window_get_root_layer(main_data.window);
  GRect window_bounds = layer_get_bounds(window_root);
  window_stack_push(main_data.window, true);
  // initialize menu layer
  prv_initialize_menu_layer(window_root, window_bounds);
}

// Entry point
int main(void) {
```

```
  prv_initialize();
  app_event_loop();
}
```

```
//! @file drawing.h
//! @brief Main drawing code
//!
//! Contains all the drawing code for this app.
//!
//! @author Eric D. Phillips
//! @date November 28, 2015
//! @bugs No known bugs

#pragma once
#include <pebble.h>

//! Render a MenuLayer cell
//! @param ctx The cell's drawing context
void drawing_render_cell(GContext *ctx);
```

```
// @file drawing.c
// @brief Main drawing code
//
// Contains all the drawing code for this app.
//
// @author Eric D. Phillips
// @date November 28, 2015
// @bugs No known bugs

#include "drawing.h"

// Clock cell constants
#define CELL_CLOCK_TICK_WIDTH 3
#define CELL_CLOCK_TICK_LENGTH 8
#define CELL_CLOCK_HR_HAND_INSET 37
#define CELL_CLOCK_MIN_HAND_INSET 23


static void prv_print_mem(void *ptr, size_t length) {
  printf("Start print [%d]...", (int)length);
  for (uint8_t *byte = ptr; byte < (uint8_t*)ptr + length; byte++) {
    printf("%d", *byte);
  }
}

// Render rich text with different fonts
// Arguments are specified as two char* arrays, one of text and the other of fonts
static void prv_render_rich_text(GContext *ctx, GRect bounds, char **text, char **font) {
  // print arguments
  printf("Bounds: %d %d %d %d", bounds.origin.x, bounds.origin.y, bounds.size.w,
    bounds.size.h);
  printf("Text: %s", text[0]);
  printf("Font: %s", font[0]);
  psleep(200);
  // draw simple geometry to test if GContext is valid
  graphics_fill_circle(ctx, grect_center_point(&bounds), 30);
```

```
  graphics_draw_rect(ctx, grect_inset(bounds, GEdgeInsets1(10)));
  printf("Simple Draw Complete");
  psleep(200);
  // create text draw args
  char my_buff[] = "Hello";
  GRect my_bounds = GRect(0, 0, 144, 50);
  GFont my_font = fonts_get_system_font(FONT_KEY_GOTHIC_14);
  printf("Attributes Created");
  psleep(200);
  // draw text
  graphics_draw_text(ctx, my_buff, my_font, my_bounds, GTextOverflowModeFill, GTextAlignmentCenter,
    NULL);   // <-- Crashes here
  printf("Text Draw Complete");
  psleep(200);
}


// -------- THIS FUNCTION IS NEVER CALLED ---------- //
// However, changing anything here still prevents the crash
static void uncalled_function_1(void) {
  // get time
  time_t t_time;
  t_time = time(NULL);
  tm tm_time;
  tm_time = *localtime(&t_time);
  printf("Clock 1");
// check draw mode
  char digit_buff[6], symbol_buff[3], date_buff[16];
  // get text
  strftime(digit_buff, sizeof(digit_buff), "%l:%M", &tm_time);
  strftime(symbol_buff, sizeof(symbol_buff), "%p", &tm_time);
  strftime(date_buff, sizeof(date_buff), "%a, %b %e", &tm_time);
  // draw text
  char *txt[2];
  txt[0] = digit_buff;
  txt[1] = symbol_buff;
  char *font[2];
  font[0] = FONT_KEY_GOTHIC_18_BOLD;
  font[1] = FONT_KEY_GOTHIC_18_BOLD;
  prv_render_rich_text(NULL, GRectZero, txt, font);
  char *txt_1[5];
  txt_1[0] = date_buff;
  char *font_1[5];
  font_1[0] = FONT_KEY_GOTHIC_18_BOLD;
  prv_render_rich_text(NULL, GRectZero, txt_1, font_1);
  // draw tick marks
  GRect tick_bounds;
  tick_bounds = grect_inset(GRectZero, GEdgeInsets1(-15));
  graphics_context_set_stroke_width(NULL, CELL_CLOCK_TICK_WIDTH);
  graphics_context_set_stroke_color(NULL, GColorWhite);
  int32_t angle;
  for (angle = 0; angle < TRIG_MAX_ANGLE; angle += TRIG_MAX_ANGLE / 12) {
    graphics_draw_line(NULL, gpoint_from_polar(tick_bounds, GOvalScaleModeFillCircle, angle),
      grect_center_point(&tick_bounds));
  }
  graphics_fill_rect(NULL, grect_inset(GRectZero, GEdgeInsets1(CELL_CLOCK_TICK_LENGTH)), 0,
    GCornerNone);
  printf("Clock 3");
  // draw date
  GRect date_bounds;
  date_bounds = GRectZero;
  date_bounds.origin.y += date_bounds.size.h / 2 - 12;
  date_bounds.origin.x += date_bounds.size.w * 2 / 3;
  date_bounds.size.w /= 4;
  strftime(date_buff, sizeof(date_buff), "%e", &tm_time);
  graphics_draw_text(NULL, date_buff, fonts_get_system_font(FONT_KEY_GOTHIC_18_BOLD), date_bounds,
    GTextOverflowModeFill, GTextAlignmentCenter, NULL);
```

```c
  // calculate hands
  GPoint hr_point = gpoint_from_polar(grect_inset(GRectZero, GEdgeInsets1(CELL_CLOCK_HR_HAND_INSET)),
    GOvalScaleModeFillCircle,
    (tm_time.tm_hour % 12 * 60 + tm_time.tm_min) * TRIG_MAX_ANGLE / (1440 / 2));
  GPoint min_point = gpoint_from_polar(grect_inset(GRectZero,
      GEdgeInsets1(CELL_CLOCK_MIN_HAND_INSET)), GOvalScaleModeFillCircle,
    tm_time.tm_min * TRIG_MAX_ANGLE / 60);
  // draw hands
  graphics_draw_line(NULL, grect_center_point(&GRectZero), GPointZero);
  graphics_draw_line(NULL, grect_center_point(&GRectZero), GPointZero);
}


// Render a MenuLayer cell
void drawing_render_cell(GContext *ctx) {
  // get cell parameters
  char *txt[5];
  txt[0] = "Hello";
  txt[1] = "World";
  char *font[5];
  font[0] = FONT_KEY_GOTHIC_18_BOLD;
  font[1] = FONT_KEY_GOTHIC_18_BOLD;
  // render cell
  prv_render_rich_text(ctx, GRect(0, 0, 144, 50), txt, font);  // <-- Enters here


  // --------- FROM HERE ON NEVER GETS CALLED ---------- //
  // However, changing anything here still prevents the crash
  if (rand() < 0) {
    uncalled_function_1();
  }
}
```