Daily Planner

Kai-Jun Huang, Chih-Hua Zhang, Wei-Tsung Wang, Yu-Ming Xu

Abstract

Most of the to-do list applications on the market don't have a weekly plan function. Our main interface is developed in C with GTK, and the database is managed with C and SQLite. We have developed an application that offers both a daily planner and a weekly planner. The app helps people plan their daily and weekly study and life tasks more clearly, allowing them to keep track of all tasks during busy semesters.
Source Code (Github): **https://github.com/IMFAat/C-program**

Keywords: GTK ; Database; SQLite; C ;

1. Introduction

Motivation:

The reason we chose this topic stems from our observation that many to-do applications on the market primarily focus on daily scheduling while giving less attention to weekly planning. This often makes it difficult for users to allocate time for long-term or multiple tasks, especially when available time is limited. Therefore, we aim to provide a more comprehensive solution that alleviates users' concerns about having too little time for planning, ultimately improving both work efficiency and quality of life.

Our goal is to create a to-do list app that combines both daily and weekly planning, enabling users to clearly organize their tasks for the entire week and avoid time pressure or the risk of missing important items. Ultimately, such a tool will make it easier for users to plan their time, enhancing both work predictability and effectiveness.

Contribution:

Table 1. Work Distribution Chart

Members	Work Distribution
Chih-Hua Zhang	UI, Style and Animation Data processing (Frontend) Comment (Frontend)
Kai-Jun Huang	UI, Style and Animation Data processing (Frontend) Comment (Frontend)
Wei-Tsung Wang	SQL API (CRUD) Comment (SQLite) Translation
Yu-Ming Xu	Comment (SQLite) Document and Slide

All authors have read and agreed to the published version of the manuscript.

2.Materials and Methods

Table 2. Tools and Their Purposes for App Development

Tools	Applications
Windows	Build and run Windows apps
macOS	Build and run macOS apps
Visual Studio Code	Code editor/IDE
CLion	Code editor/IDE
CMake	Build system generator
C	Program controlling
GTK	GUI and animations framework
SQLite	Lightweight database
XML	UI structure markup
CSS	UI styling language

2.1 Materials

In this project, we use C for development and divide the project into two main components: the frontend and the database API. For the frontend, we utilize the GTK toolkit to implement UI/UX design and animations. On the database side, we use the SQLite library to perform CRUD (Create, Read, Update, Delete) operations, packaging these functionalities into APIs for seamless integration with the frontend.

2.1.1 Introduction of GTK

GTK, short for the GIMP Toolkit, is an open-source, cross-platform toolkit designed for creating graphical user interfaces (GUIs). Originally developed by the GNOME project, GTK has become a widely used foundation for building desktop applications. It leverages object-oriented programming principles through the GObject library, which enables object orientation in C, the programming language used to implement GTK(Clasen, Bassi et al. 2024).

2.1.2 Introduction of SQLite

SQLite is an open-source C library that provides a lightweight, fast, self-contained, highly reliable, and full-featured SQL database engine. Due to its cross-platform compatibility and lightweight nature, SQLite has been widely adopted in numerous projects, including Android and macOS. It is regarded as the most widely used database engine in the world.(Hipp 2023)

2.2 Structure

2.2.1 File Structure

```
Daily Planner
├── CMakeLists.txt
├── README.md
├── Screens
│   └── MainScreen
│       ├── MainScreen.c
│       └── MainScreen.h
├── main.c
├── sqlite
│   ├── daily
│   │   ├── daily.c
│   │   └── daily.h
│   └── weekly
│       ├── weekly.c
│       └── weekly.h
└── src
    ├── Icons
    │   ├── menu.png
    │   └── trash.png
    └── MainScreen
        ├── MainScreen.css
        └── MainScreen.ui
```

- **CmakeLists.txt**

  The CMakeLists.txt file is the configuration file for CMake, used to specify the build process for a project. CMake is a cross-platform build tool that generates platform-specific build files (such as Makefiles or Visual Studio project files) based on the instructions in the CMakeLists.txt. The contents of this file include settings like the project name, source files to be compiled, compiler options, and dependencies.

- **main.c**

  The main.c file serves as the entry point of the program, responsible for defining the core structure of the application, including the initialization of the Application and the main Window. In this file, the UI layout for the main screen (MainScreen) and its associated CSS styles are loaded, ensuring smooth functionality and a cohesive visual design for the application.

- **MainScreen.c / MainScreen.h**

  MainScreen.h is one of the header files included in main.c, primarily serving the purpose of exposing the MainScreen function from MainScreen.c to main.c. The file MainScreen.c implements all the features required by the main interface (MainScreen) of the application, including animations, data processing, and interactions with the database.

- **MainScreen.css / MainScreen.ui**

  MainScreen.ui serves as the main UI for Main Screen, written in XML format and loaded in main.c via GtkBuilder. It defines most of the components used in Main Screen, such as buttons of the drawer, menus for adding daily and weekly activities, and more. Meanwhile, MainScreen.css primarily handles the styling for Main Screen, including specifications for border radius, margins, background color, and other visual design elements.

- **daily.c / daily.h weekly.c / weekly.h**

These files primarily provide database APIs related to daily and weekly activities, including functionalities for creating, deleting, retrieving, and updating records. It facilitates seamless integration and interaction between the front-end and the database.

2.2.2 Structure of frontend

In this project, we primarily use the GtkBuilder XML format to design our user interface (UI). This approach allows us to define UI components declaratively in XML, which are then dynamically loaded into the application through a series of GTK APIs. This separation of UI design from code simplifies the development process, enabling modularity and easier maintenance.

For styling, we use CSS to customize the appearance of UI elements. By modifying attributes such as border-radius, background color, and other properties, we create a visually appealing and user-friendly interface. This flexibility in design not only enhances aesthetics but also improves the overall usability of the application.

2.2.3 Code Snippet and Explanation of MainScreen Function

In main.c, we load MainScreen.ui and use the following code to load MainScreen.css into GtkCssProvider, then add it to the default display to apply the styles.

```c
// Load ui from file
GtkBuilder *mainBuilder = gtk_builder_new_from_file(ROOT_PATH"/src/MainScreen/MainScreen.ui");

// Find MainScreen widget from main builder(ui file)
GtkWidget *main_screen = GTK_WIDGET(gtk_builder_get_object(mainBuilder, "MainScreen"));

// Define the css file path
const char cssPath[]=ROOT_PATH"/src/MainScreen/MainScreen.css";
GtkCssProvider * cssProvider = gtk_css_provider_new();
// Load css from path
gtk_css_provider_load_from_path(cssProvider, cssPath);
// Add css to display
gtk_style_context_add_provider_for_display(gdk_display_get_default(),GTK_STYLE_PROVIDER(cssProvider),GTK_STYLE_PROVIDER_PRIORITY_USER);
```

After loading the UI file, we call the MainScreen function from MainScreen.c via MainScreen.h to handle the main screen. We then query all the activities for the day using Daily.c and Weekly.c and display them on the screen.

```c
// Initialize database
d_initialize_database();
w_initialize_database();
// Load today daily activities
d_load_entries(g_reg_daily, now->tm_year + 1900, now->tm_mon + 1, now->tm_mday);
// Add daily activities to window
for (int i = 0; i < g_reg_daily->len; i++)
{
    Daily_Mission_Data_widget *d_regis = &g_array_index(g_reg_daily, Daily_Mission_Data_widget, i);
    v_D_create_widget(d_regis);
    gtk_box_append((GtkBox *)combine->daily_added_box, (GtkWidget *)d_regis->Display_Box);
}
// Load this week weekly activities
w_load(g_reg_weekly, now->tm_year + 1900, i_now_week);
// Add weekly activities to window
for (int i = 0; i < g_reg_weekly->len; i++)
{
    Weekly_Mission_Data_widget *w_regis = &g_array_index(g_reg_weekly, Weekly_Mission_Data_widget, i);
    v_W_create_widget(w_regis);
    gtk_box_append((GtkBox *)combine->weekly_added_box, (GtkWidget *)w_regis->Display_Box);
}
```

2.2.4 Drawer Animation

In this section, I will introduce our drawer animation. First, we locate the Drawer Button from GtkBuilder using its ID and connect it to the clicked signal.

```c
// Find Drawer Button by ID
GtkWidget *drawerButton = (GtkWidget *) gtk_builder_get_object (builder, "DrawerButton");
// Connect clicked signal to drawer button
g_signal_connect  (drawerButton, "clicked", G_CALLBACK (DrawerButtonOnClick),  combine);

// Set animation (10ms)
g_timeout_add(10, (GSourceFunc)drawerAnimation, data);
```

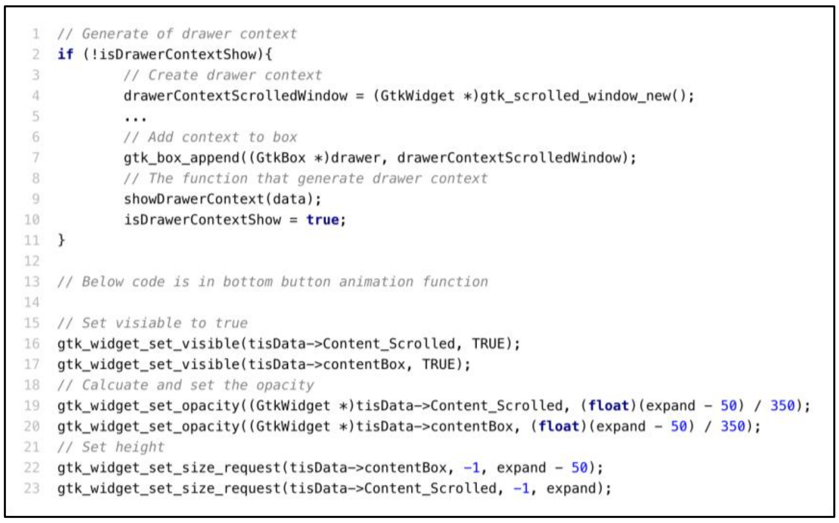
Next, we proceed with handling the animation. We set GTK to run the animation function every 10 milliseconds, adjusting the drawer's width by expanding or shrinking 20px based on its open or close state. The process continues until the width reaches 330px or falls below 65px, at which point the drawer's width is set to 330px or 65px.

When the drawer is first opened, we generate its content, including a label and a calendar. As the drawer's closing animation reaches the final step, we remove the content from the drawer box. Simultaneously, as the width increases or decreases, the transparency of the drawer's content is adjusted through the ongoing animation.

```c
// Width of each increase/reduction
int dWidth = 20 * (isDrawerOpen ? -1 : 1);
// Get drawer width
int width = gtk_widget_get_width(drawer);
// Increase/reduce the width
width += dWidth;
// If width is over or below
if (width > 330) width = 330;
if (width < 65) width = 65;
// Set width
gtk_widget_set_size_request(drawer, width, -1);

// Calculate the opacity of drawer context
double drawerOpacity = (width - 65) * 1. / (330 - 65);
// Set opacity
gtk_widget_set_opacity(drawerContextScrolledWindow, drawerOpacity);

// If it is the last step of animation, remove the context
if (!isDrawerOpen && step == 0){
    gtk_box_remove((GtkBox *)drawer, drawerContextScrolledWindow);
}
```

2.2.5 Bottom Button Animation (Add Button Animation)

The principle behind the Bottom Button Animation is like that of the Drawer Animation, with the key difference being that the Drawer adjusts its width, while the Bottom Button Animation adjusts its height. Additionally, the display method for the contents of the Bottom Button (menus for adding daily and weekly activities) differs from that of the Drawer. In the case of the Drawer, a new Drawer Context is created each time the drawer is opened, and when the drawer is closed, the context is removed from the Box and the memory is freed. On the other hand, the contents of the Bottom Button are written directly in the .ui file. At the start of the program, the contents are set to be invisible, with an opacity of 0 and the height compressed to a minimum. As the program runs, the contents become visible, the height gradually increases, and the opacity transitions from 0.0 to 1.0. When the height is minimized again, the opacity is set to 0, and the contents are made invisible, thus achieving the animation effect.



```c
// Generate of drawer context
if (!isDrawerContextShow){
        // Create drawer context
        drawerContextScrolledWindow = (GtkWidget *)gtk_scrolled_window_new();
        ...
        // Add context to box
        gtk_box_append((GtkBox *)drawer, drawerContextScrolledWindow);
        // The function that generate drawer context
        showDrawerContext(data);
        isDrawerContextShow = true;
}

// Below code is in bottom button animation function

// Set visiable to true
gtk_widget_set_visible(tisData->Content_Scrolled, TRUE);
gtk_widget_set_visible(tisData->contentBox, TRUE);
// Calcuate and set the opacity
gtk_widget_set_opacity((GtkWidget *)tisData->Content_Scrolled, (float)(expand - 50) / 350);
gtk_widget_set_opacity((GtkWidget *)tisData->contentBox, (float)(expand - 50) / 350);
// Set height
gtk_widget_set_size_request(tisData->contentBox, -1, expand - 50);
gtk_widget_set_size_request(tisData->Content_Scrolled, -1, expand);
```

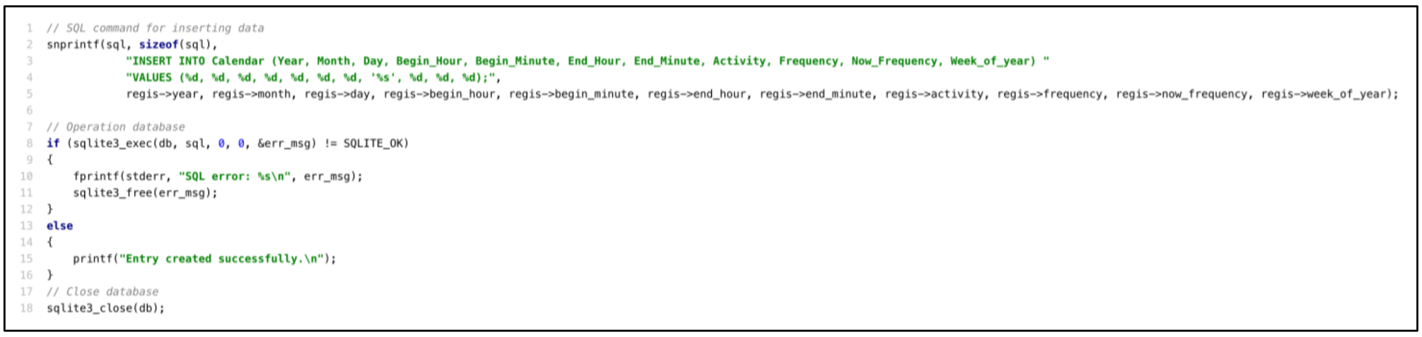
2.2.6 Combine Gtk with SQL

In the frontend, the main data is transmitted using structs and GArrays.

The primary struct we used is Mission_Combination, which contains two GArrays, components to be read, containers to add components, and data for drawer animations.

GArray is a data type provided by Glib, similar to the vector in C++. The two GArrays store weekly and daily data, as well as components associated with them, such as (GtkButton)Add_frequency and (GtkWidget)Display_Box.

For the backend integration, the GArrays are passed to the API, and the values are stored in the GArrays.



```c
// SQL command for inserting data
snprintf(sql, sizeof(sql),
            "INSERT INTO Calendar (Year, Month, Day, Begin_Hour, Begin_Minute, End_Hour, End_Minute, Activity, Frequency, Now_Frequency, Week_of_year) "
            "VALUES (%d, %d, %d, %d, %d, %d, %d, '%s', %d, %d, %d);",
            regis->year, regis->month, regis->day, regis->begin_hour, regis->begin_minute, regis->end_hour, regis->end_minute, regis->activity, regis->frequency, regis->now_frequency, regis->week_of_year);

// Operation database
if (sqlite3_exec(db, sql, 0, 0, &err_msg) != SQLITE_OK)
{
    fprintf(stderr, "SQL error: %s\n", err_msg);
    sqlite3_free(err_msg);
}
else
{
    printf("Entry created successfully.\n");
}
// Close database
sqlite3_close(db);
```

2.2.7 Re-connect signal

Event triggers are mainly declared using the g_signal_connect(instance, detailed_signal, c_handler, data) function.

The instance in the function is the object (pointer type) that triggers the event. Since the object to be triggered is found through GArray, and GArray dynamically allocates memory, when the length of the GArray is updated, the addresses of the members within the GArray may change. This can cause the previously declared pointer address to change, potentially leading to the instance parameter becoming a null pointer, which results in undefined behavior. Therefore, when modifying the GArray, all bound events are destroyed using the v_signal_destroy function, and then reconnected using the v_signal_connect function.

```c
// Re-connect the signal of daily
for (int i = 0; i < a_d_regis->len; i++)
{
    d_regis = &g_array_index(a_d_regis, Daily_Mission_Data_widget, i);
    g_signal_connect(d_regis->Add_frequency, "clicked", G_CALLBACK(v_D_update_widget), d_regis);
    g_signal_connect(d_regis->RemoveButton, "clicked", G_CALLBACK(v_D_remove_widget), data);
    g_signal_connect(d_regis->Add_frequency, "clicked", G_CALLBACK(v_sync), data);
    g_signal_connect(d_regis->revise, "clicked", G_CALLBACK(v_D_revise), data);
}
// Re-connect the signal of weakly
for (int i = 0; i < a_w_regis->len; i++)
{
    w_regis = &g_array_index(a_w_regis, Weekly_Mission_Data_widget, i);
    g_signal_connect(w_regis->Add_frequency, "clicked", G_CALLBACK(v_W_update_widget), w_regis);
    g_signal_connect(w_regis->RemoveButton, "clicked", G_CALLBACK(v_W_remove_widget), data);
    g_signal_connect(w_regis->revise, "clicked", G_CALLBACK(v_W_revise), data);
}
```

2.2.8 The button of process +1

The event triggered by (GtkButton *)Add_frequency is v_W_update_widget (for daily and weekly plans). This event compares the address of the triggered button with the addresses of all buttons in the GArray. It finds the button with the matching address, retrieves the data of the corresponding GArray member, increments the now_frequency (current progress), and simultaneously updates the visibility bar.

```c
// Get curent frequency and plus 1
regis->now_frequency++;

// Set progress bar
gtk_progress_bar_set_fraction((GtkProgressBar *)processBar, regis->now_frequency * 1. / regis->frequency);

// Set frequency label
char frequencyText[frequencyLength * 2 + 4];
sprintf(frequencyText, "%d / %d", regis->now_frequency, regis->frequency);
gtk_label_set_label((GtkLabel *)frequencyLabel, frequencyText);
```

2.2.9 The button of edit activity

The event triggered by (GtkButton *)revise is v_D_revise (for daily plans) and v_W_revise (for weekly plans). When triggered, a new task window pops up, but the button label changes from "ADD" to "REVISE". The function determines whether to execute v_daily_add(v_weekly_add) or v_D_revise_btn(v_W_revise_btn) based on the text on the button. At the same time, the event parameters are passed to the components in the new task window to facilitate editing.

In the v_D_revise_btn(v_W_revise_btn) function, the updated values are validated and then passed into the GArray. The values are also updated in the database, and the component is refreshed. Finally, the window is reset and closed.

```c
// Set the label of ADD button to REVISE
gtk_button_set_label((GtkButton *)SD_widget->Add, "REVISE");
// Get the activity data from the array
for (int i = 0; i < d_gdata->len; i++)
{
    d_regis = &g_array_index(d_gdata, Daily_Mission_Data_widget, i);
    if (button == d_regis->revise)
    {
        break;
    }
}
// Set the data to the widget
gtk_editable_set_text((GtkEditable *)SD_widget->Activity, d_regis->activity);
gtk_widget_set_sensitive((GtkWidget *)SD_widget->Activity, false);
gtk_spin_button_set_value((GtkSpinButton *)SD_widget->Frequency, d_regis->frequency);
gtk_spin_button_set_value((GtkSpinButton *)SD_widget->Begin_hour, d_regis->begin_hour);
gtk_spin_button_set_value((GtkSpinButton *)SD_widget->Begin_minute, d_regis->begin_minute);
gtk_spin_button_set_value((GtkSpinButton *)SD_widget->End_hour, d_regis->end_hour);
gtk_spin_button_set_value((GtkSpinButton *)SD_widget->End_hour, d_regis->end_hour);
```

```c
// Get the data from the array
for (int i = 0; i < d_gdata->len; i++)
{
    d_regis = &g_array_index(d_gdata, Daily_Mission_Data_widget, i);
    if (!strcmp(d_regis->activity, gtk_editable_get_text((GtkEditable *)combine->daily_widget->Activity)))
    {
        break;
    }
}
// Set the data from the widget
d_regis->frequency = gtk_spin_button_get_value(combine->daily_widget->Frequency);
d_regis->begin_hour = gtk_spin_button_get_value(combine->daily_widget->Begin_hour);
d_regis->begin_minute = gtk_spin_button_get_value(combine->daily_widget->Begin_minute);
d_regis->end_hour = gtk_spin_button_get_value(combine->daily_widget->End_hour);
d_regis->end_minute = gtk_spin_button_get_value(combine->daily_widget->End_minute);
d_regis->now_frequency--;

// Update database
v_D_update_widget(button, d_regis);
// Set the label of ADD button to ADD and clear the widget
gtk_button_set_label((GtkButton *)combine->daily_widget->Add, "ADD");
gtk_widget_set_sensitive((GtkWidget *)combine->daily_widget->Activity, true);
gtk_editable_set_text((GtkEditable *)combine->daily_widget->Activity, "");
gtk_spin_button_set_value(combine->daily_widget->Frequency, 1);
gtk_spin_button_set_value(combine->daily_widget->Begin_hour, 0);
gtk_spin_button_set_value(combine->daily_widget->Begin_minute, 0);
gtk_spin_button_set_value(combine->daily_widget->End_hour, 0);
gtk_spin_button_set_value(combine->daily_widget->End_minute, 0);
```

2.2.10 The button remove of activity

The "Remove" button is primarily executed through the v_D_remove and v_W_remove functions. During the execution process, the address of the pressed "Remove" button is compared with all the "Remove" buttons in the GArray. The matching GArray member is identified, and then the associated GTK components are deleted. Afterward, the corresponding data in the database is removed, and finally, the GArray member is deleted.

```c
// Remove the data from the box
int removeIndex = 0;
for (int i = 0; i < g_data->len; i++)
{
    Daily_Mission_Data_widget *d = &g_array_index(g_data, Daily_Mission_Data_widget, i);
    if (d->Display_Box == d_regis->Display_Box)
    {
        gtk_box_remove((GtkBox *)combine->daily_added_box, (GtkWidget *)d->Display_Box);
        removeIndex = i;
        break;
    }
}
// Remove the data from the array
g_array_remove_index(g_data, removeIndex);



// Set the label of ADD button to REVISE
gtk_button_set_label((GtkButton *)SD_widget->Add, "REVISE");
// Get the activity data from the array
for (int i = 0; i < d_gdata->len; i++)
{
    d_regis = &g_array_index(d_gdata, Daily_Mission_Data_widget, i);
    if (button == d_regis->revise)
    {
        break;
    }
}
```

2.2.11 The select day event of calendar

In the select_day event of the calendar, all the components on the screen are cleared, and the two GArrays are emptied. Based on the selected date, the calculate_iso_week_number function is used to calculate the current week number. Then, the relevant data is extracted from the database and placed into the GArrays. Using the functions v_D_create_widget and v_W_create_widget, the components are recreated, but all the components' sensitive property is set to false.

```c
// Get selected date from calendar
int week = calculate_iso_week_number(select_year, select_month, select_day);

int calculate_iso_week_number(int year, int month, int day)
{
    // Set the time info
    struct tm timeinfo = {0};
    timeinfo.tm_year = year - 1900;
    timeinfo.tm_mon = month - 1;
    timeinfo.tm_mday = day;
    timeinfo.tm_hour = 12;

    // Get week of year
    char buffer[3];
    strftime(buffer, sizeof(buffer), "%W", &timeinfo);
    return atoi(buffer);
}
```

2.2.12 SQL

The SQL functionality is divided into two parts: daily and weekly, with a total of six functions. These functions are initialize_database, create_entry, read_entries, update_entry, delete_entry, and load_entry.

initialize_database: This function loads and initializes the database.

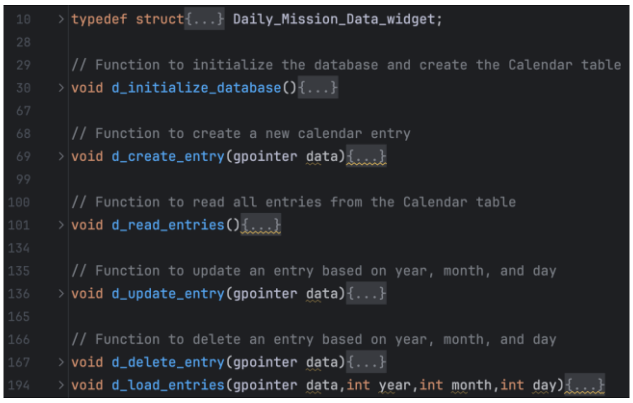
create_entry: This function stores the values from the provided GArray into the database based on certain conditions.

read_entries: This function outputs all the data to the terminal.

update_entry: This function updates the values in the database based on the provided GArray.

delete_entry: This function deletes rows of data in the database based on the provided GArray.

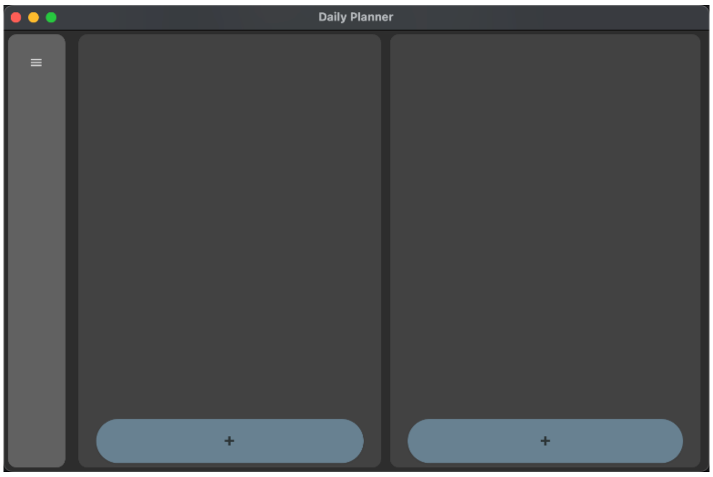
load_entry: During initialization, this function transfers all relevant data for the current period into the GArray.



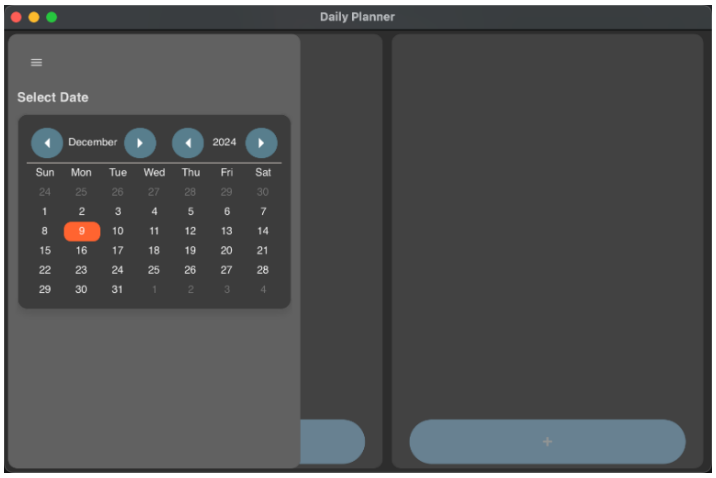
```c
typedef struct{...} Daily_Mission_Data_widget;

// Function to initialize the database and create the Calendar table
void d_initialize_database(){...}

// Function to create a new calendar entry
void d_create_entry(gpointer data){...}

// Function to read all entries from the Calendar table
void d_read_entries(){...}

// Function to update an entry based on year, month, and day
void d_update_entry(gpointer data){...}

// Function to delete an entry based on year, month, and day
void d_delete_entry(gpointer data){...}
void d_load_entries(gpointer data,int year,int month,int day){...}
```

3. Results

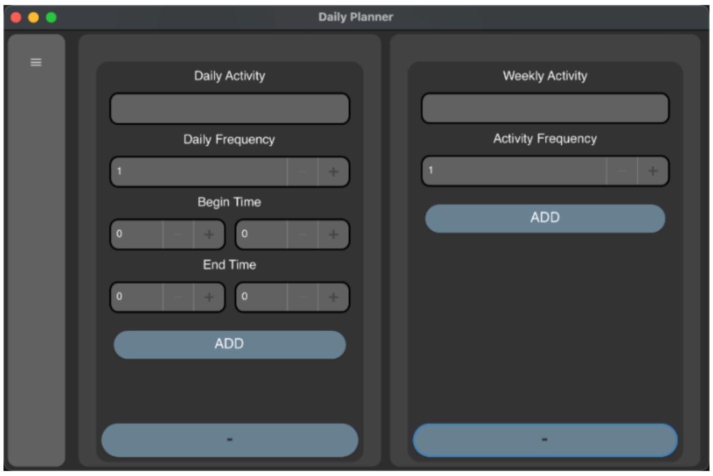
Github Repositories: https://github.com/IMFAat/C-program



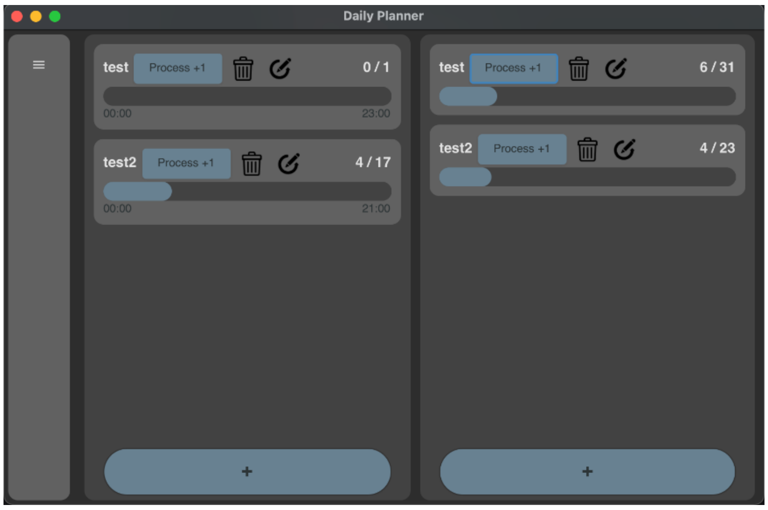
(Main screen): This is the main screen of our Daily Planner app.



(Drawer): The drawer contains a Calendar, allowing users to select their target date.



(Daily and Weekly text field): These text boxes allow users to input data.



(Activity): Display the data entered by the user.

4. Discussion

The "Daily Planner" we developed is not just a typical to-do application; it offers significantly more potential and value. Compared to traditional to-do list apps, what sets 'Daily Planner' apart is its simple yet non-burdensome UI design, which prioritizes user experience and practicality, ensuring that every feature is both intuitive and convenient for users.

In addition, Daily Planner features a unique weekly planning function, offering a significant advantage over similar applications. With this feature, users can easily organize their tasks and goals for the entire week, improving time management and reducing the frustration of getting lost in everyday tasks. This design makes Daily Planner not only ideal for daily task management but also helps users plan for longer timeframes—whether for work, study, or personal life—ensuring that everything remains organized and on track.

We believe that it will bring significant benefits to users, playing a crucial role in improving work efficiency, helping users achieve their goals, and reducing stress. Furthermore, this application not only promises a more organized life for individuals but also has the potential to become a valuable tool for business and team management, providing substantial economic benefits.

5. Conflicts of Interest: The authors declare no conflict of interest.

6. References:

Clasen, M., et al. (2024). "GNOME / gtk · GitLab." Retrieved 12/09, 2024, from https://gitlab.gnome.org/GNOME/gtk/.
Hipp, D. R. (2023). "About SQLite." from https://www.sqlite.org/about.html.