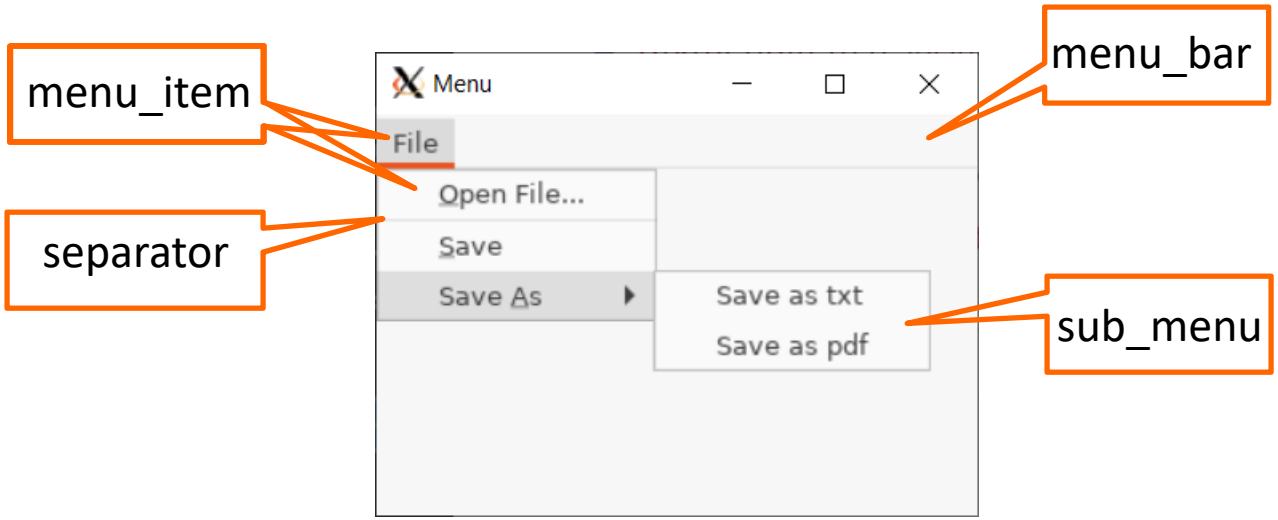
CSE216 Programming Abstractions
X Window Programming

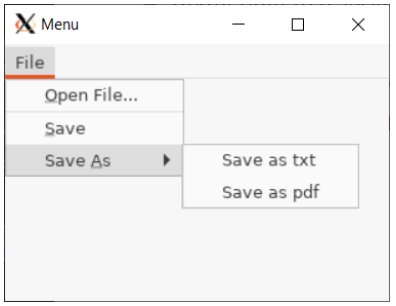
YoungMin Kwon

Menu

- Menu
  - A set of options provided to the user to find information and execute functions



Menu



- GtkMenuShell
  - Base class of GtkMenu and GtkMenuBar

```
void gtk_menu_shell_append(GtkMenuShell *menu_shell,
                           GtkWidget    *child);
```

- GtkMenuBar, GtkMenu

```
GtkWidget* gtk_menu_bar_new(void);
GtkWidget* gtk_menu_new(void);
```

- GtkMenu

```
GtkWidget* gtk_menu_item_new(void);
GtkWidget* gtk_menu_item_new_with_label(const gchar *label);
GtkWidget* gtk_menu_item_new_with_mnemonic(const gchar *label);
void       gtk_menu_item_set_submenu(GtkMenuItem *menu_item,
                                     GtkWidget   *submenu);
```

Menu.c

- Tasks
  - Add menus and submenus to the application
  - Add callbacks to the menu items

```c
//menu.c

#include <stdio.h>
#include <gtk/gtk.h>
void on_delete(GtkWidget *widget, gpointer data) {
    printf("bye.\n");
    gtk_main_quit();
}
void on_activate_menu(GtkWidget *widget, gpointer data) {
    printf("data: %s\n", (char*)data);
}

static GtkWidget *make_menu() {
    //menu bar
    GtkWidget *menu_bar = gtk_menu_bar_new();

    //submenu
    GtkWidget *menu_head = gtk_menu_item_new_with_mnemonic("_File");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu_bar), menu_head);
    GtkWidget *menu = gtk_menu_new();
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(menu_head), menu);
```

```
//menu item
GtkWidget *menu_item = gtk_menu_item_new_with_mnemonic("_Open File...");
gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);
g_signal_connect(G_OBJECT(menu_item), "activate",
    G_CALLBACK(on_activate_menu), "open");

//separator
menu_item = gtk_separator_menu_item_new();
gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);

//menu item
menu_item = gtk_menu_item_new_with_mnemonic("_Save");
gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);
g_signal_connect(G_OBJECT(menu_item), "activate",
    G_CALLBACK(on_activate_menu), "save");
```

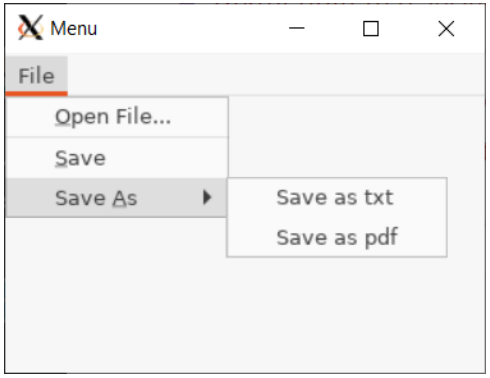
```
    //submenu
    GtkWidget *submenu_head = gtk_menu_item_new_with_mnemonic("Save _As");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), submenu_head);
    menu = gtk_menu_new();
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(submenu_head), menu);

    //menu item
    menu_item = gtk_menu_item_new_with_label("Save as txt");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);
    g_signal_connect(G_OBJECT(menu_item), "activate",
        G_CALLBACK(on_activate_menu), "save as txt");

    //menu item
    menu_item = gtk_menu_item_new_with_label("Save as pdf");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);
    g_signal_connect(G_OBJECT(menu_item), "activate",
        G_CALLBACK(on_activate_menu), "save as pdf");

    return menu_bar;
}
```

```c
int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);

    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Menu");
    gtk_widget_set_size_request(GTK_WIDGET(window), 300, 200);
    g_signal_connect(G_OBJECT(window), "delete_event",
        G_CALLBACK(on_delete), NULL);

    GtkWidget *vbox = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
    gtk_container_add(GTK_CONTAINER(window), vbox);

    GtkWidget *menu_bar = make_menu();
    gtk_box_pack_start(GTK_BOX(vbox), menu_bar, FALSE, FALSE, 0);

    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

To compile and execute

- Add lines for menu to Makefile

```
$ make menu
$ ./menu
```



Dialog

- Modal window
  - There may be times when an application cannot proceed without the user's input
  - A modal window is the only window in the application that receives events (key, mouse, …)
    - It forces the user to make a decision

Dialog.c

- Tasks
  - Make a Modal Dialog
  - Handle the responses

GtkDialog

```
GtkWidget* gtk_dialog_new(void);
GtkWidget* gtk_dialog_new_with_buttons(
                const gchar     *title,
                GtkWindow       *parent,
                GtkDialogFlags   flags,
                const gchar     *first_button_text,…) G_GNUC_NULL_TERMINATED;
typedef enum {
  GTK_DIALOG_MODAL = 1 << 0,
  GTK_DIALOG_DESTROY_WITH_PARENT = 1 << 1, …
} GtkDialogFlags;
typedef enum {
  GTK_RESPONSE_NONE         = -1,
  GTK_RESPONSE_REJECT       = -2,
  GTK_RESPONSE_ACCEPT       = -3,
  GTK_RESPONSE_DELETE_EVENT = -4,
  GTK_RESPONSE_OK           = -5,
  GTK_RESPONSE_CANCEL       = -6,
  GTK_RESPONSE_CLOSE        = -7,
  GTK_RESPONSE_YES          = -8,
  GTK_RESPONSE_NO           = -9,
  GTK_RESPONSE_APPLY        = -10,
  GTK_RESPONSE_HELP         = -11
} GtkResponseType;
```

GtkDialog

```
//get the response from a modal dialog
gint gtk_dialog_run(GtkDialog *dialog);

//get the content area of a dialog
GtkWidget *gtk_dialog_get_content_area(GtkDialog *dialog);

//get the button widget for the response
GtkWidget* gtk_dialog_get_widget_for_response(GtkDialog *dialog,
                                              gint response_id);

//get the response for the button widget
gint gtk_dialog_get_response_for_widget(GtkDialog *dialog,
                                        GtkWidget *widget);

//make window a modal dialog
void gtk_window_set_modal(GtkWindow *window, gboolean modal);
```

```c
//dialog.c
#include <stdio.h>
#include <gtk/gtk.h>

GtkWidget *toplevel_win;

//forward declaration
int show_dialog(char *title, char *message, GtkWidget *parent);

void on_delete(GtkWidget *widget, gpointer data) {
    int res = show_dialog("Exit", "Do you want to exit?", toplevel_win);
    if(res == GTK_RESPONSE_OK) {
        printf("bye.\n");
        gtk_main_quit();
    }
}
void on_activate_open(GtkWidget *widget, gpointer data) {
    int res = show_dialog("Open", "Do you want to open?", toplevel_win);
    printf("Open: %s\n", res == GTK_RESPONSE_OK ? "yes" : "no");
}
void on_activate_save(GtkWidget *widget, gpointer data) {
    int res = show_dialog("Save", "Do you want to save?", toplevel_win);
    printf("Save: %s\n", res == GTK_RESPONSE_OK ? "yes" : "no");
}
```

```c
int show_dialog(char *title, char *message, GtkWidget *parent) {
    //make a modal dialog
    GtkWidget *dlg = gtk_dialog_new_with_buttons(
                        title,                  //title
                        GTK_WINDOW(parent),     //parent
                        GTK_DIALOG_MODAL|GTK_DIALOG_DESTROY_WITH_PARENT,//flags
                        "_Cancel", GTK_RESPONSE_CANCEL, //button text, resp
                        "_OK",     GTK_RESPONSE_OK,     //button text, resp
                        NULL);
    //add a message to the dialog
    GtkWidget *content = gtk_dialog_get_content_area(GTK_DIALOG(dlg));
    GtkWidget *label = gtk_label_new(message);
    gtk_container_add(GTK_CONTAINER(content), label);
    gtk_widget_show(label);

    //set the default response
    gtk_dialog_set_default_response(GTK_DIALOG(dlg), GTK_RESPONSE_OK);

    //wait for the modal dialog to return
    int res = gtk_dialog_run(GTK_DIALOG(dlg));

    //destroy the dialog
    gtk_widget_destroy(dlg);

    return res;
}
```

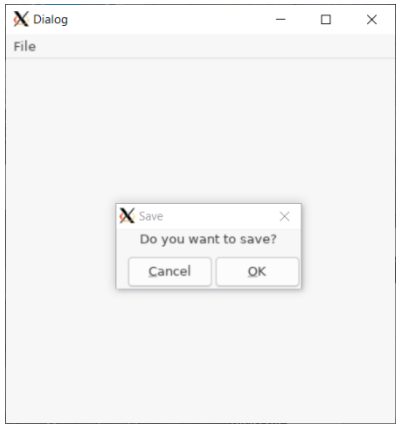
```
GtkWidget *make_menu() {
    //menu bar
    GtkWidget *menu_bar = gtk_menu_bar_new();

    //submenu
    GtkWidget *menu_head = gtk_menu_item_new_with_mnemonic("_File");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu_bar), menu_head);
    GtkWidget *menu = gtk_menu_new();
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(menu_head), menu);

    //menu item
    GtkWidget *menu_item = gtk_menu_item_new_with_mnemonic("_Open");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);
    g_signal_connect(G_OBJECT(menu_item), "activate",
        G_CALLBACK(on_activate_open), NULL);

    //menu item
    menu_item = gtk_menu_item_new_with_mnemonic("_Save");
    gtk_menu_shell_append(GTK_MENU_SHELL(menu), menu_item);
    g_signal_connect(G_OBJECT(menu_item), "activate",
        G_CALLBACK(on_activate_save), NULL);

    return menu_bar;
}
```

```c
int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);

    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Text");
    gtk_widget_set_size_request(GTK_WIDGET(window), 400, 400);
    g_signal_connect(G_OBJECT(window), "delete_event",
        G_CALLBACK(on_delete), NULL);

    toplevel_win = window;

    GtkWidget *vbox = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
    gtk_container_add(GTK_CONTAINER(window), vbox);

    GtkWidget *menu_bar = make_menu();
    gtk_box_pack_start(GTK_BOX(vbox), menu_bar, FALSE, FALSE, 0);

    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

To compile and execute

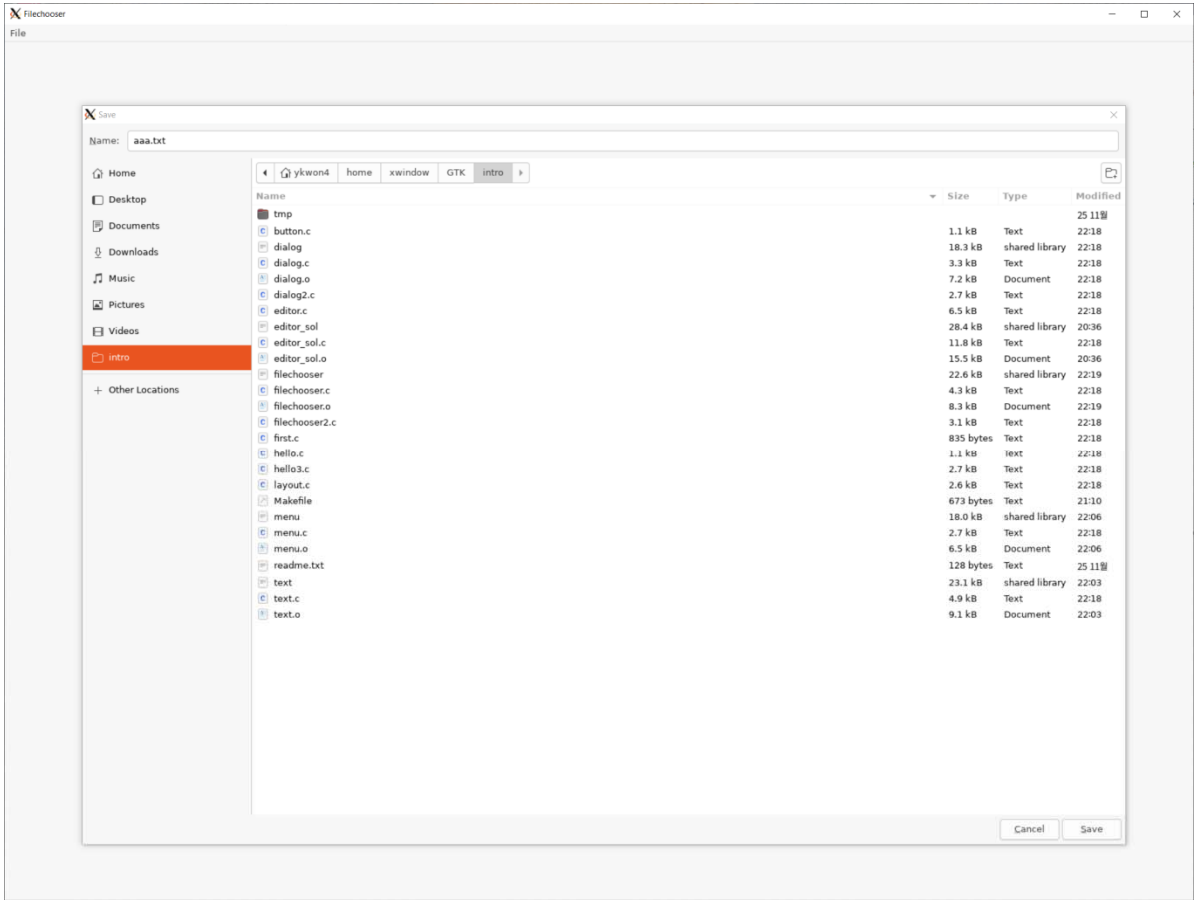
- Add lines for dialog to Makefile

```
$ make dialog
$ ./dialog
```



FileChooserDialog

- FileChooserDialog
  - A dialog box suitable for File/Open or File/Save as



FileChooserDialog

- GtkFileChooserDialog

```
GtkWidget *gtk_file_chooser_dialog_new(
              const gchar           *title,
              GtkWindow             *parent,
              GtkFileChooserAction   action,
              const gchar           *first_button_text,
              …) G_GNUC_NULL_TERMINATED;

typedef enum {
  GTK_FILE_CHOOSER_ACTION_OPEN,
  GTK_FILE_CHOOSER_ACTION_SAVE,
  GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER,
  GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER
} GtkFileChooserAction;

gchar*   gtk_file_chooser_get_filename(GtkFileChooser *chooser);
gboolean gtk_file_chooser_set_filename(GtkFileChooser *chooser,
                                       const char     *filename);
```

FileChooser.c

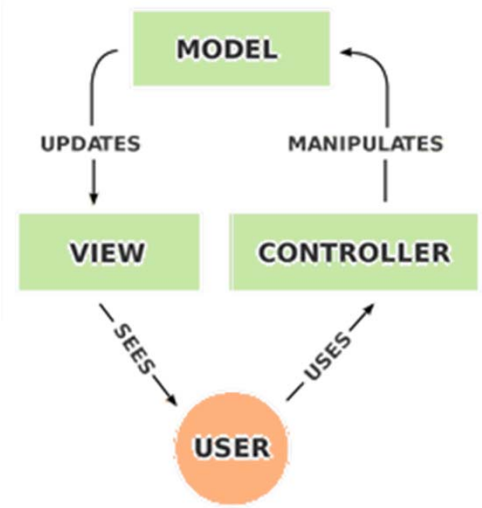
- Tasks
  - Make a GtkFileChooserDialog
  - Handle the responses

```
//filechooser.c

//TODO: copy dialog.c to filechooser.c
//TODO: update on_activate_open and on_activate_save functions
…

void on_activate_open(GtkWidget *widget, gpointer data) {
    GtkWidget *dlg = gtk_file_chooser_dialog_new("Open",
                         GTK_WINDOW(toplevel_win),
                         GTK_FILE_CHOOSER_ACTION_OPEN,
                         "_Cancel", GTK_RESPONSE_CANCEL,
                         "_Open",   GTK_RESPONSE_ACCEPT,
                         NULL);
    gtk_dialog_set_default_response(GTK_DIALOG(dlg), GTK_RESPONSE_ACCEPT);

    if(gtk_dialog_run(GTK_DIALOG(dlg)) == GTK_RESPONSE_ACCEPT) {
        char *fname = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dlg));
        printf("open: %s\n", fname);
        g_free(fname);
    }

    gtk_widget_destroy(dlg);
}
```

```c
void on_activate_save(GtkWidget *widget, gpointer data) {
    GtkWidget *dlg = gtk_file_chooser_dialog_new("Save",
                         GTK_WINDOW(toplevel_win),
                         GTK_FILE_CHOOSER_ACTION_SAVE,
                         "_Cancel", GTK_RESPONSE_CANCEL,
                         "_Save",   GTK_RESPONSE_ACCEPT,
                         NULL);
    gtk_dialog_set_default_response(GTK_DIALOG(dlg), GTK_RESPONSE_ACCEPT);

    if(gtk_dialog_run(GTK_DIALOG(dlg)) == GTK_RESPONSE_ACCEPT) {
        char *fname = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dlg));
        printf("save: %s\n", fname);
        g_free(fname);
    }

    gtk_widget_destroy(dlg);
}
```

MVC Design Pattern

- Model-View-Controller (MVC) design pattern
  - Model stores the data
  - View shows the data
  - Controller updates the data



Text

- GtkTextView and GtkTextBuffer
  - GtkTextBuffer is a model with the text data
  - GtkTextView is a view and a controller for the model
  - One or more TextViews can share a TextBuffer

```
//make a text view
GtkWidget *    gtk_text_view_new            (void);

//make a text view with a textbuffer
GtkWidget *    gtk_text_view_new_with_buffer(GtkTextBuffer *buffer);

//set/get text buffer to/from text view
void           gtk_text_view_set_buffer     (GtkTextView   *text_view,
                                             GtkTextBuffer *buffer);
GtkTextBuffer *gtk_text_view_get_buffer     (GtkTextView   *text_view);
```

```
// text buffer

// get/set text
gchar *gtk_text_buffer_get_text (GtkTextBuffer     *buffer,
                                 const GtkTextIter *start,
                                 const GtkTextIter *end,
                                 gboolean           include_hidden_chars);
void gtk_text_buffer_set_text   (GtkTextBuffer *buffer,
                                 const gchar   *text,
                                 gint           len);

// get/set modified
gboolean gtk_text_buffer_get_modified (GtkTextBuffer *buffer);
void     gtk_text_buffer_set_modified (GtkTextBuffer *buffer,
                                       gboolean       setting);

// cut/copy/paste clipboard
void gtk_text_buffer_cut_clipboard   (GtkTextBuffer *buffer,
                                       GtkClipboard  *clipboard,
                                       gboolean       default_editable);
void gtk_text_buffer_copy_clipboard  (GtkTextBuffer *buffer,
                                       GtkClipboard  *clipboard);
void gtk_text_buffer_paste_clipboard (GtkTextBuffer *buffer,
                                       GtkClipboard  *clipboard,
                                       GtkTextIter   *override_location,
                                       gboolean       default_editable);
```

Text

- Iterators (GtkTextIter)
  - An iterator represents a position between two chars in the TextBuffer
  - Iterators will be invalidated after modifying the TextBuffer
- Scroll
  - To scroll, add TextView to GtkScrolledWindow

```
//make iterators for the start/end positions of the buffer
void gtk_text_buffer_get_start_iter(GtkTextBuffer *buffer,
                                    GtkTextIter   *iter);
void gtk_text_buffer_get_end_iter  (GtkTextBuffer *buffer,
                                    GtkTextIter   *iter);
//make a scrolled window
//adjustment: define lower, upper bounds, step size, page size, …
GtkWidget* gtk_scrolled_window_new (GtkAdjustment *hadjustment,
                                    GtkAdjustment *vadjustment);
```

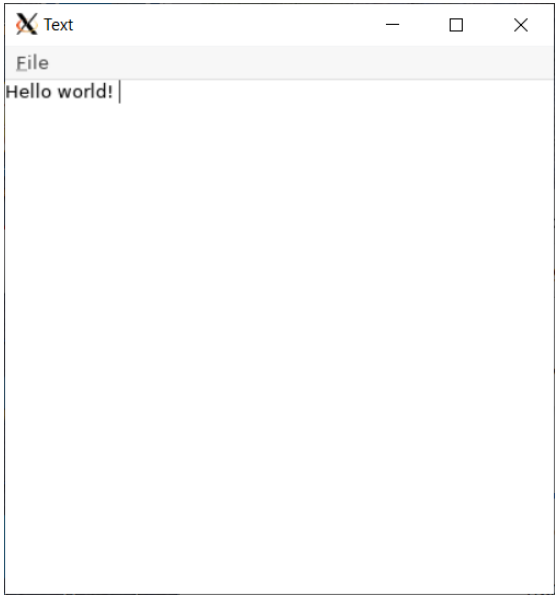
```c
//text.c
//copy dialog.c to text.c

#include <stdio.h>
#include <gtk/gtk.h>

//global text view and text buffer
GtkWidget     *text_view;
GtkTextBuffer *text_buf;
GtkWidget     *toplevel_win;

int show_dialog(char *title, char *message, GtkWidget *parent);

void on_delete(GtkWidget *widget, gpointer data) {
    int res = show_dialog("Exit", "Do you want to exit?", toplevel_win);
    if(res == GTK_RESPONSE_OK) {
        printf("bye.\n");
        gtk_main_quit();
    }
}
```

```
void on_activate_new(GtkWidget *widget, gpointer data) {
    int proceed = 1;

    //check if text_buf is modified
    if(gtk_text_buffer_get_modified(text_buf)) {
        int res = show_dialog("Modified", "Do you wan to clear text?",
                              toplevel_win);
        proceed = (res == GTK_RESPONSE_OK);
    }

    if(proceed) {
        //disable user interaction with text_view
        gtk_widget_set_sensitive(text_view, FALSE);

        //update text_buf
        gtk_text_buffer_set_text(text_buf, "", 0);

        //clear the modified flag
        gtk_text_buffer_set_modified(text_buf, FALSE);

        //enable user interaction with text_view
        gtk_widget_set_sensitive(text_view, TRUE);
    }
}
```

```
void on_activate_save(GtkWidget *widget, gpointer data) {
    //disable user interaction with text_view
    gtk_widget_set_sensitive (text_view, FALSE);

    //get the start and end positions of text_buf
    GtkTextIter start, end;
    gtk_text_buffer_get_start_iter(text_buf, &start);
    gtk_text_buffer_get_end_iter  (text_buf, &end);

    //get the contents of text_buf and print it
    gchar *contents = gtk_text_buffer_get_text(
                          text_buf, &start, &end, FALSE);
    printf("%s\n", contents);
    g_free(contents);

    //clear the modified flag
    gtk_text_buffer_set_modified(text_buf, FALSE);
    //enable user interaction with text_view
    gtk_widget_set_sensitive(text_view, TRUE);
}
```

```
GtkWidget *make_text_win() {
    //make a scrolled window
    GtkWidget *scroll_win = gtk_scrolled_window_new(NULL, NULL);

    //make a text view and add it to scrolled window
    text_view = gtk_text_view_new();
    gtk_container_add(GTK_CONTAINER(scroll_win), text_view);

    //get the text buffer from text_view
    text_buf = gtk_text_view_get_buffer(GTK_TEXT_VIEW(text_view));

    return scroll_win;
}

int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);

    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Text");
    gtk_widget_set_size_request(GTK_WIDGET(window), 400, 400);
    g_signal_connect(G_OBJECT(window), "delete_event",
        G_CALLBACK(on_delete), NULL);

    toplevel_win = window;
…
```

```
    GtkWidget *vbox = gtk_box_new(GTK_ORIENTATION_VERTICAL, 0);
    gtk_container_add(GTK_CONTAINER(window), vbox);

    GtkWidget *menu_bar = make_menu(window);
    gtk_box_pack_start(GTK_BOX(vbox), menu_bar, FALSE, FALSE, 0);

    //make text window
    GtkWidget *text_win = make_text_win();
    gtk_box_pack_start(GTK_BOX(vbox), text_win, TRUE, TRUE, 0);

    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```



Exercise

- Make a text editor
  - Download editor.c
  - Implement TODOs
  - Make the editor behaves like the video clip

Thank you for your attention during the semester!

Any questions or comments?

- Please submit your Course Evaluation at

  https://p22.courseval.net/etw/ets/et.asp?nxappid=SU2&nxmid=start

  https://stonybrook.campuslabs.com/eval-home/

SUNY Korea