Introduction to Graphical User Interface Programming – GTK+

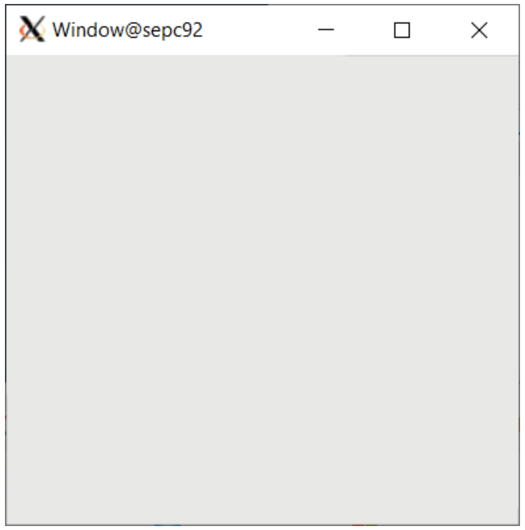
Backgound

- GTK+ supports the development of graphical user interface (GUI) in Linux.
- GTK+ was adopted as the default graphical toolkit of GNOME and XFCE, two of the most popular Linux desktop environments.
- GTK+ is written entirely in C, and the majority of GTK+ software is also written in C.
- GTK+ is built on top of a number of other libraries such as ATK, Pango, etc.

Simple Example

Aim

- We will write a simple GTK+ program using GTK 3.0, which shows the basic structure with a GUI on Linux.
- Below is a screenshot of the program



- This program will create an empty 200 × 200 pixel window.

Simple Example

Code

```
/*   example-0.c   */
#include <gtk/gtk.h>
static void activate (GtkApplication* app, gpointer user_data) {
   GtkWidget *window;
   window = gtk_application_window_new (app);
   gtk_window_set_title (GTK_WINDOW (window), "Window");
   gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);
   gtk_widget_show_all (window);
}
int main (int argc, char **argv) {
   GtkApplication *app;
   int status;
   app = gtk_application_new ("org.gtk.example", G_APPLICATION_FLAGS_NONE);
   g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
   status = g_application_run (G_APPLICATION (app), argc, argv);
   g_object_unref (app);
   return status;       }
```

Simple Example

Description

- The <gtk/gtk.h> file includes all of the widgets, variables, functions, and structures available in GTK+ as well as header files from other libraries that GTK+ depends on.
  - When you implement applications using GTK+, it is enough to use only <gtk/gtk.h>, except for some advanced applications.
- A GTK+ program consists of widgets.
  - Components such as windows, buttons, and scrollbars are called widgets.

Simple Example

Description

- Purpose of the main() function is to create a GtkApplication object and run it. In this example a GtkApplication instance is created and initialized using gtk_application_new().
- When creating a GtkApplication you need to pick an application identifier (a name) and input to gtk_application_new() as parameter.
    - E.g., org.gtk.example is used but for choosing an identifier
- Lastly gtk_application_new() takes a GApplicationFlags argument, which control some of the capabilities that your application has, like being able to open files specified on the command line, or parsing command line options.
- Next the activate signal is connected to the activate() function above the main() functions. The activate signal will be sent when your application is launched with g_application_run() on the line below.

Simple Example

Description

- The gtk_application_run() also takes as arguments the pointers to the command line arguments counter and string array; this allows GTK to parse specific command line arguments that control the behavior of GTK itself. The parsed arguments will be removed from the array, leaving the unrecognized ones for your application to parse.
- Within g_application_run the activate() signal is sent and we then proceed into the activate() function of the application.
- In the activate() function we want to construct our GTK window, so that a window is shown when the application is launched.
- The call to gtk_application_window_new() will create a new GtkApplicationWindow instance and store it inside the window pointer. The window will have a frame, a title bar, and window controls depending on the platform.

Simple Example

Description

- A window title is set using gtk_window_set_title(). This function takes a GtkWindow pointer and a string as input. As our window pointer is a GtkWidget pointer, we need to cast it to GtkWindow. But instead of casting window via the usual C cast operator (GtkWindow *), window should be cast using the GTK_WINDOW() macro.
- Finally the window size is set using gtk_window_set_default_size() and the window is then shown by GTK via gtk_widget_show_all().
- If you exit window by pressing the X, the g_application_run() in the main loop returns with a number which is saved inside an integer named “status”. Afterwards, the GtkApplication object is freed from memory with g_object_unref(). Finally the status integer is returned to the operating system, and the GTK application exits.

Simple Example

Compile

- The following command compiles the code

```
gcc  -o example-0 example-0.c `pkg-config --cflags --libs gtk+-3.0`
```

- Take care to type backticks
- In addition to the gcc compiler, you need to use the pkg-config application, which returns a list of specified libraries or paths.
    - pkg-config --cflags, returns directory names to the compiler's 'include path'. This will make sure that the GTK+ header files are available to the compiler.
    - pkg-config --libs gtk+-3.0, appends options to the command line used by the linker including library directory path extensions and a list of libraries needed for linking to the executable.

Events, Signals, and Callbacks

Basic information

- *Events* represent some activities to which we may want to respond
- For example, we may want our program to perform some action when the following occurs:
  - the mouse is moved
  - the mouse is dragged
  - a mouse button is clicked
  - a graphical button is clicked
  - a keyboard key is pressed
  - a timer expires
- Events often correspond to user actions.

Events, Signals, and Callbacks

Basic information

- GTK+ is a system that relies on events, signals, and callbacks.
    - An event is a message emitted by the X Window System such as clicking mouse or typing a keyboard etc. it is emitted and sent to your application to be interpreted by the signal system provided by GLib.
    - A signal is reaction to an event, which is emitted by a GtkObject.
    - You can tell GTK+ to run a function when the signal is emitted. This is called a callback function.
- The callback function will be called when the action has occurred and the signal is emitted or when you have explicitly emitted the signal.

Events, Signals, and Callbacks

Basic information

- The g_signal_connect() function connects the signal.

```
gulong g_signal_connect ( gpointer object,
                          const gchar *signal_name,
                          G_Callback handler,
                          gpointer data);
```

- The object is the widget that is to be monitored for the signal.
- You specify the name of the signal you want to keep track of with the signal_name.
- The handler is the callback function that will be called when the signal is emitted, cast with G_CALLBACK().
- The data allows you to send a pointer to the callback function.
- The return value of g_signal_connect() is the handler identifier of the signal.

Events, Signals, and Callbacks

Callback functions

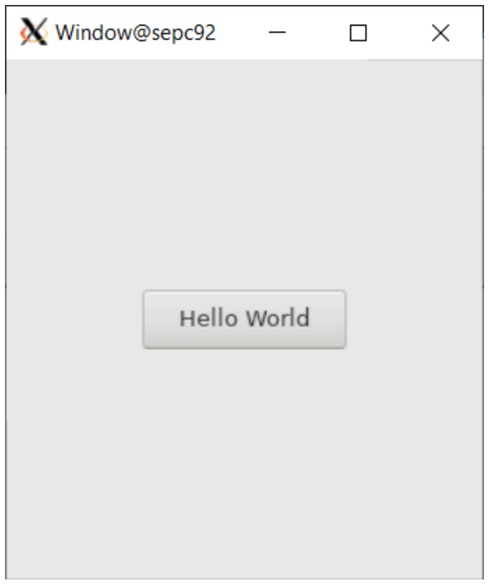
- Callback functions specified in g_signal_connect() will be called when the signal is emitted on the widget to which it was connected. The callback functions are in the following form and are named by a programmer.

```
static void callback_function ( GtkApplication* app,
                                ... /* other possible arguments */ ... ,
                                gpointer data);
```

- The app is the object from g_signal_connect().
- There are other possible arguments that may appear in the middle as well, although this is not always the case.
- The data correspond to the last argument of g_signal_connect(), which is gpointer data. Since the data is passed as a void pointer, you can replace the data type with what you want to cast.

Adding a Button

- We will add a button to our window, with the label “Hello World”.



- When the button is clicked, the program will display the text “Hello World” on the terminal.

Adding a Button

Code

```
/*   example-1.c   */
#include <gtk/gtk.h>
static void print_hello (GtkWidget *widget, gpointer data) {
   g_print ("Hello World\n");
}
static void activate (GtkApplication *app, gpointer user_data) {
   GtkWidget *window;
   GtkWidget *button;
   GtkWidget *button_box;
   window = gtk_application_window_new (app);
   gtk_window_set_title (GTK_WINDOW (window), "Window");
   gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);
   button_box = gtk_button_box_new (GTK_ORIENTATION_HORIZONTAL);
   gtk_container_add (GTK_CONTAINER (window), button_box);
   button = gtk_button_new_with_label ("Hello World");
   g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);
   g_signal_connect_swapped (button, "clicked", G_CALLBACK (gtk_widget_destroy), window);
   gtk_container_add (GTK_CONTAINER (button_box), button);
   gtk_widget_show_all (window);    }
```

Adding a Button
Code

```
/*   example-1.c  continue  */
Int main (int argc, char **argv)  {
   GtkApplication *app;
   int status;

   app = gtk_application_new ("org.gtk.example", G_APPLICATION_FLAGS_NONE);
   g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
   status = g_application_run (G_APPLICATION (app), argc, argv);
   g_object_unref (app);

   return status;    }
```

- A new GtkWidget pointer is declared to accomplish this, *button*, and is initialized by calling gtk_button_new_with_label(), which returns a GtkButton to be stored inside *button*. Afterwards *button* is added to our window.

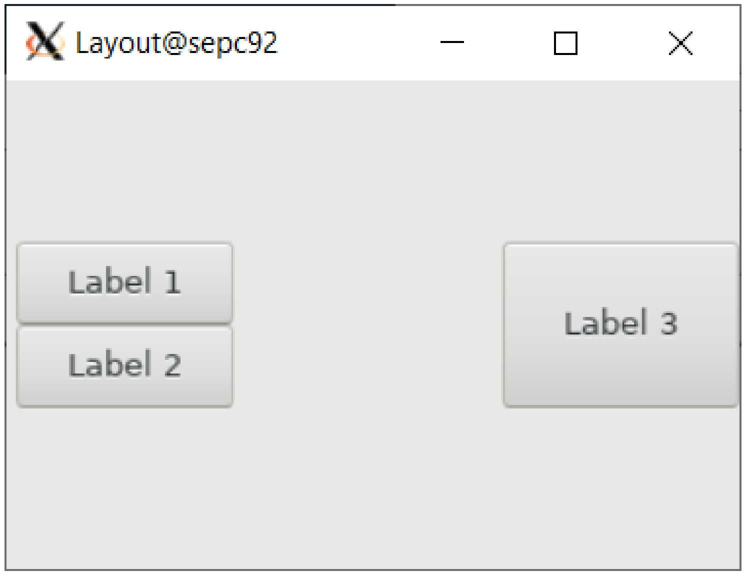
Widgets
Basic information

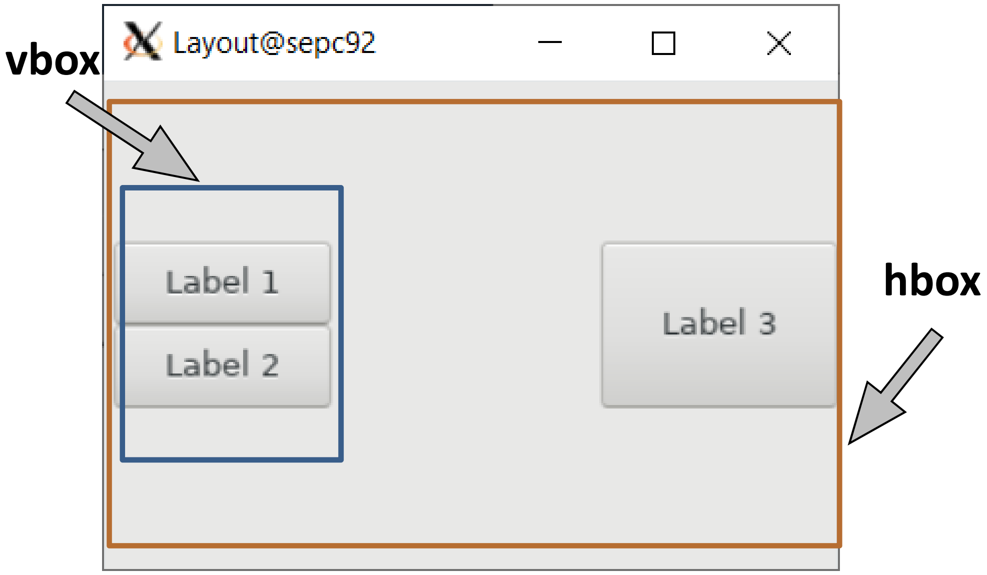
- Widgets are the basic building blocks of a GUI application.
- There are some common widgets such as window widget, container (layout) widget, label widget, button widget, single-line entry widget, etc
- The main purpose of a container class is to allow a parent widget to contain one or more children.

Widgets

Packing Widgets

- In this example, we will implement the application which contains hbox and vbox widgets.





- When Label 1 is clicked, it will display the text “Hello World” on the terminal.

Packing Widgets

Principle

- When creating an application, you'll want to put more than one button inside a window.
- when you want to put more than one widget into a window, how do you control where that widget is positioned ? This is where packing comes in.
- These are invisible widget *containers* that we can pack our widgets into and come in two forms, a horizontal box, and a vertical box.
- When packing widgets into a horizontal box, the objects are inserted horizontally from left to right or right to left depending on the call used.
- In a vertical box, widgets are packed from top to bottom or vice versa. You may use any combination of boxes inside or beside other boxes to create the desired effect.

Packing Widgets

Principle

- The gtk_box_pack_start() and gtk_box_pack_end() functions are used to place objects inside of these containers.
- The gtk_box_pack_start() function will start at the top and work its way down in a vbox, and pack left to right in an hbox.
- The gtk_box_pack_end() will do the opposite, packing from bottom to top in a vbox, and right to left in an hbox. Using these functions allow us to right justify or left justify our widgets and may be mixed in any way to achieve the desired effect.

Widgets

Code

```
/*   layout.c   */
#include <gtk/gtk.h>
static void print_hello (GtkWidget *widget, gpointer   data) {
  g_print ("Hello World\n");
}
static void activate (GtkApplication *app, gpointer user_data) {
   GtkWidget *window;
   GtkWidget *label1, *label2, *label3;
   GtkWidget *hbox;
   GtkWidget *vbox;
   window = gtk_application_window_new (app);
   gtk_window_set_title (GTK_WINDOW (window), "Layout");
   gtk_window_set_default_size (GTK_WINDOW (window), 300, 200);
   label1 = gtk_button_new_with_label("Label 1");
   g_signal_connect (label1,"clicked", G_CALLBACK(print_hello),NULL);
   label2 = gtk_button_new_with_label("Label 2");
   label3 = gtk_button_new_with_label("Label 3");
   hbox = gtk_button_box_new(GTK_ORIENTATION_HORIZONTAL);
   vbox = gtk_button_box_new(GTK_ORIENTATION_VERTICAL);
   gtk_box_pack_start(GTK_BOX(vbox), label1, TRUE, FALSE, 5);
   gtk_box_pack_start(GTK_BOX(vbox), label2, TRUE, FALSE, 5);
   gtk_box_pack_start(GTK_BOX(hbox), vbox, FALSE, FALSE, 5);
   gtk_box_pack_start(GTK_BOX(hbox), label3, FALSE, FALSE, 5);
   gtk_container_add(GTK_CONTAINER(window), hbox);
   gtk_widget_show_all(window);
}
```

Widgets

Code

```
/*   layout.c  continue  */

Int  main (int argc, char **argv)
{
   GtkApplication *app;
   int status;

   app = gtk_application_new ("org.gtk.example", G_APPLICATION_FLAGS_NONE);
   g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
   status = g_application_run (G_APPLICATION (app), argc, argv);
   g_object_unref (app);
   return status;   }
```

Widgets

Description of layout

- We have three labels and one vbox and one hbox in the program.
  - The vbox contains the label1 and the lable2.
  - The hbox contains the vbox which consists of label1 and label2 and the label3.
  - Finally, the hbox is located in the window

Widgets

Description of layout

- This function adds child to box, packed with reference to the start of box.

```
void gtk_box_pack_start (GtkBox *box, GtkWidget *child,
                         gboolean expand, gboolean fill, guint padding);
```

- The first parameter should be box object where children will be packed.
- The second one means a child widget to be added to box.
- These three parameters, expand, fill, and padding, are related to spacing of children.
- Examples

```
gtk_box_pack_start(GTK_BOX(vbox), label1, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(vbox), label2, TRUE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), vbox, FALSE, FALSE, 5);
gtk_box_pack_start(GTK_BOX(hbox), label3, FALSE, FALSE, 5);
```

Widgets

Description of layout

- This function adds widget to container.

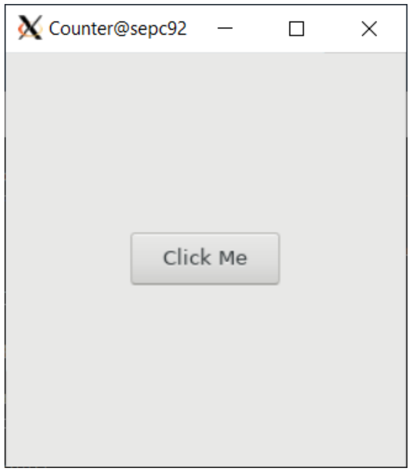
```
gtk_container_add(GTK_CONTAINER(window), hbox);
```

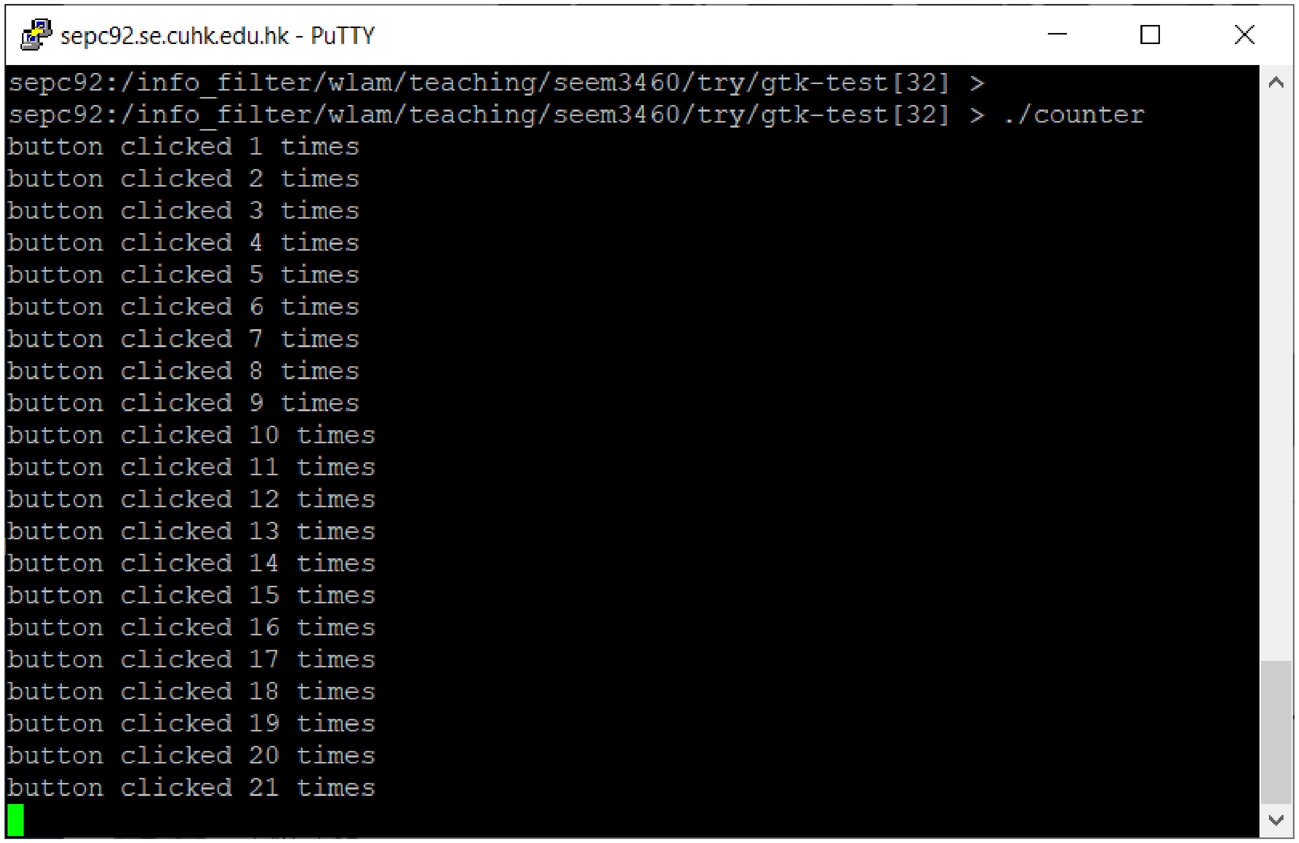
- The first parameter is the container to be added
- The second one is the widget.

Widgets

Example of Basic widgets (1)

- In this example, we will show the usage of button widget.





```
sepc92:/info_filter/wlam/teaching/seem3460/try/gtk-test[32] >
sepc92:/info_filter/wlam/teaching/seem3460/try/gtk-test[32] > ./counter
button clicked 1 times
button clicked 2 times
button clicked 3 times
button clicked 4 times
button clicked 5 times
button clicked 6 times
button clicked 7 times
button clicked 8 times
button clicked 9 times
button clicked 10 times
button clicked 11 times
button clicked 12 times
button clicked 13 times
button clicked 14 times
button clicked 15 times
button clicked 16 times
button clicked 17 times
button clicked 18 times
button clicked 19 times
button clicked 20 times
button clicked 21 times
```

Widgets

Code of basic widgets (1) – counter.c

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

static int counter=0;
static void greet (GtkWidget *widget, gpointer   data)   {
   g_print("%s clicked %d times\n",(char*)data,++counter);    }

static void activate (GtkApplication *app, gpointer user_data)   {
   GtkWidget *window;
   GtkWidget *button;
   GtkWidget *button_box;
   window = gtk_application_window_new (app);
   gtk_window_set_title (GTK_WINDOW (window), "Counter");
   gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);
   button_box = gtk_button_box_new (GTK_ORIENTATION_HORIZONTAL);
   gtk_container_add (GTK_CONTAINER (window), button_box);
   button = gtk_button_new_with_label ("Hello World");
   g_signal_connect (button, "clicked", G_CALLBACK (greet), "button");
   gtk_container_add (GTK_CONTAINER (button_box), button);
   gtk_widget_show_all (window);
}
```

Widgets

Code of basic widgets (1) – counter.c

```
int
main (int argc, char **argv)
{
  GtkApplication *app;
  int status;

  app = gtk_application_new ("org.gtk.example", G_APPLICATION_FLAGS_NONE);
  g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
  status = g_application_run (G_APPLICATION (app), argc, argv);
  g_object_unref (app);

  return status;
}
```

Widgets

Description of basic widgets (1)

- greet() function plays a role to show the reaction when you click the button.

```
void greet(GtkWidget* widget, gpointer data) {
 // printf equivalent in GTK+
  g_print("Welcome to GTK\n");
  g_print("%s clicked %d times\n", (char*)data, ++counter);
}
```

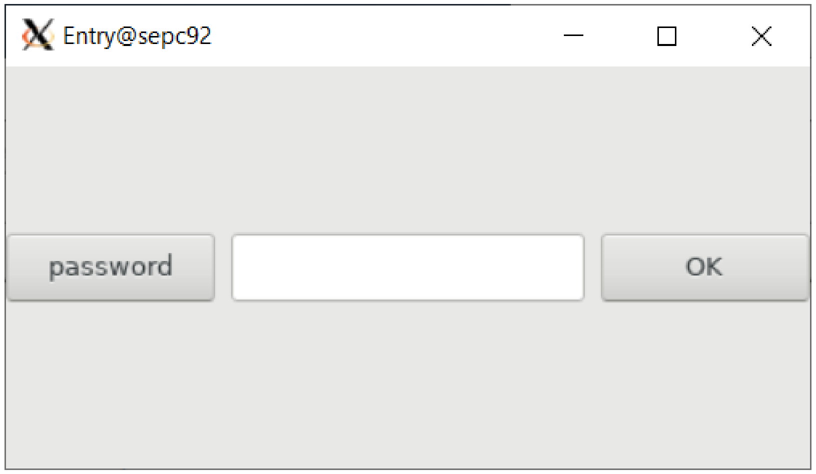
- gtk_button_new_with_label() creates a new button with a label.

```
button = gtk_button_new_with_label("Click Me!");
```

Widgets

entry

- In this example, we will show the usage of entry widget.



Widgets

Code of entry.c

```
#include <stdio.h>
#include <string.h>
const char *password = "secret";
void button_clicked(GtkWidget *widget, gpointer data) {
  const char *password_text = gtk_entry_get_text(GTK_ENTRY((GtkWidget *)data));
  if(strcmp(password_text, password) == 0)
    printf("Access granted!\n");  else      printf("Access denied!\n");      }
static void activate (GtkApplication *app, gpointer user_data) {
   GtkWidget *window;
   GtkWidget *label,*entry,*ok_button,*hbox;
   window = gtk_application_window_new (app);
   gtk_window_set_title (GTK_WINDOW (window), "Entry");
   gtk_window_set_default_size (GTK_WINDOW (window), 400, 200);
   label = gtk_button_new_with_label("password");
   entry = gtk_entry_new();
   ok_button = gtk_button_new_with_label("OK");
   g_signal_connect (ok_button,"clicked",G_CALLBACK(button_clicked),entry);
   g_signal_connect_swapped (ok_button,"clicked",G_CALLBACK(gtk_widget_destroy),window);
   hbox = gtk_button_box_new(GTK_ORIENTATION_HORIZONTAL);
   gtk_box_pack_start(GTK_BOX(hbox), label, TRUE, FALSE, 5);
   gtk_box_pack_start(GTK_BOX(hbox), entry, TRUE, FALSE, 5);
   gtk_box_pack_start(GTK_BOX(hbox), ok_button, TRUE, FALSE, 5);
   gtk_container_add(GTK_CONTAINER(window), hbox);
   gtk_widget_show_all(window);     }
```

Widgets

Code of entry.c

```
int
main (int    argc,
      char **argv)
{
  GtkApplication *app;
  int status;

  app = gtk_application_new ("org.gtk.example", G_APPLICATION_FLAGS_NONE);
  g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
  status = g_application_run (G_APPLICATION (app), argc, argv);
  g_object_unref (app);

  return status;
}
```

Widgets

Description of basic widgets (2)

- button_clicked() function plays a role to check whether or not input password is correct.

```
void button_clicked(GtkWidget *button, gpointer data)
{
const char *password_text =
gtk_entry_get_text(GTK_ENTRY((GtkWidget *)data));
 if(strcmp(password_text, password) == 0)
        printf("Access granted!\n");
else
        printf("Access denied!\n");
}
```

- gtk_label_new() function creates a new label.

```
username_label = gtk_label_new("Login: ");
```

- This function gets a text as a parameter.

Widgets

Description of basic widgets (2)

- These functions are related to entry widget.

```
password_entry = gtk_entry_new();
gtk_entry_set_visibility(GTK_ENTRY(password_entry), FALSE);
gtk_entry_get_text(GTK_ENTRY((GtkWidget *)data));
```

- gtk_entry_new() function creates a new entry widget.
- After creating the entry, we can set visibility of entry widget using gtk_entry_set_visibility() that has two parameters, entry and visible.
- Finally, we can retrieve text information from the entry widget using gtk_entry_get_text().
- gtk_button_new_with_label() creates a new button with a label.

```
ok_button = gtk_button_new_with_label("OK");
```

Useful Links

- The GTK+ Project

  http://www.gtk.org

- The GTK 3.0 Examples

  https://developer-old.gnome.org/gtk3/unstable/gtk-getting-started.html

- GObject Reference Manual

  http://library.gnome.org/devel/gobject/stable