

GTK+ / GNOME

What is GNOME/GTK+???

▶ GNOME

- GNU Network Object Model Environment
- GNOME is an attempt to make a graphical desktop environment, a visual way to present the Unix operating system
- Constantly under development
- Being a part of the GNU project, GNOME is free not only to use but also to be improved upon by anybody who wants to.

What is GNOME/GTK+?

▶ GTK+

- General Image Manipulation Program (GIMP) toolkit
- A toolkit for creating graphical user interfaces for the X Windows System
- Originally written to develop the GNU Image Manipulation Program
- Now used to develop lots of other applications, and of course, GNOME.

GTK+'s Widget Libraries

▶ GTK

- User interface widgets

▶ Glib

- Provides many useful datatypes such as hastables and arrays, macros, type conversions, string utilities, and even a lexical scanner.

▶ GDK

- Contains the wrappers for low-level windowing functions, such as shape drawing, font manipulation, and window event handling.

History of GTK+

- ▶ Spencer Kimball and Peter Mattis -- General Image Manipulation Program, or GIMP
- ▶ On January 1996, they released to the public version 0.54; users and developers alike loved it.
- ▶ for version 0.60 of GIMP released later in the year, the developers wrote their own GUI toolkit, the GIMP toolkit (GTK)
- ▶ February 1997 -- release of GIMP version 0.99 and GTK+ , an overhauled version of GTK
- ▶ Minor updates and releases for both GIMP and GTK+ followed until June 1997.
- ▶ Today, GTK+ is at version 1.2 with 2.0 on the works.

History of GNOME

- ▶ August 1997 – GNOME was announced by Miguel de Icaza
- ▶ December 1997 - GNOME 0.10 was finished; Red Hat officially supported the GNOME project
- ▶ January 1998 - created the Red Hat Advanced Development Labs
- ▶ September 1998 - single GNOME tarball was split into 4 for version 0.30: the gnome core, libraries, utilities, and media
- ▶ November 1998 - the core was already at version 0.99, and code was frozen to fix things up for the official 1.0 release.
- ▶ March 1999 - released at the Linux World Expo
- ▶ October 1999 - released version 1.0.55, which fixed a lot of bugs that users reported.
- ▶ May 2000 - version 1.2, called "Bongo GNOME"
- ▶ August 2000 - the GNOME Foundation was announced
end of March 2002 – tentative release of GNOME 2.0

Programs with GNOME and GTK+

- ▶ GNOME runs on any Unix-like platform that supports X and has GTK+ installed on it.
 - runs on FreeBSD, NetBSD, Solaris, OpenBSD, IRIX, HP-UX and AIX
 - does not run on Windows, BeOS, and MacOS
 - *GIMP, Abiword, Glade, Dia, GnuCash, Gnumeric, ETC.*

Widgets

- ▶ Widgets are abstract user interface objects used to create concrete instances of those UI objects.
- ▶ To use GTK+ :

```
#include <gtk/gtk.h>  
gtk_init (&argc, &argv);  
gtk_main ();
```

Widgets (cont'd)

- ▶ To create an object, simply declare the object type (GUI components are usually "GtkWidget"), followed by a pointer to that object.
- ▶ Then, use the appropriate method to specify and initialize which GUI component it is

e.g.

```
GtkWidget *window;
```

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
gtk_widget_show (window);
```

Widgets (cont'd)

- ▶ Here are a few more useful methods for window creation:

To set the window title:

```
void gtk_window_set_title (GtkWindow *window, const  
gchar *title);
```

To set the size:

```
void gtk_window_set_default_size (GtkWindow *window,  
gint width, gint height);
```

To set its position:

```
void gtk_window_set_position (GtkWindow *window,  
GtkWindowPosition position);
```

Note: `gint` and `gchar` are simply typedefs of integers and characters respectively.

Widgets (cont'd)

- ▶ Methods for adding widgets in the window
*void gtk_container_add (GtkContainer *container,
GtkWidget *widget);*

-- Use macro to type cast a window to a container.

`GTK_CONTAINER ()`

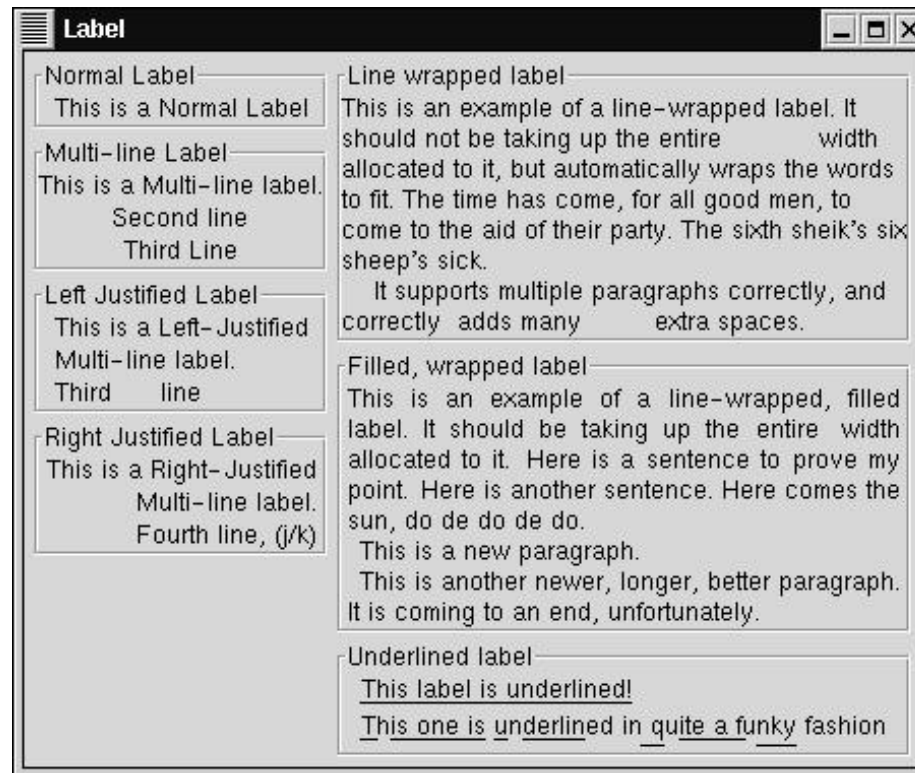
e.g

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

Other Widgets

Label

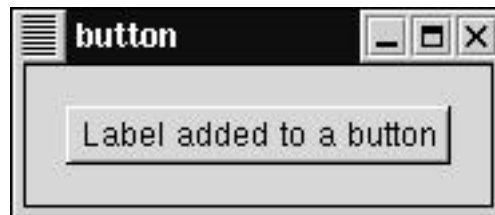
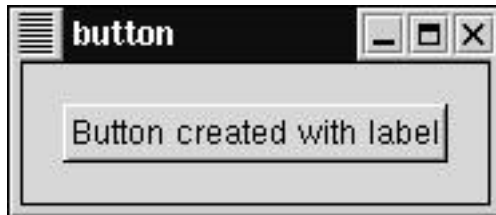
```
GtkWidget* gtk_label_new (const gchar * str);  
void gtk_label_set_justify (GtkLabel * label, GtkJustification jtype);  
void gtk_label_set_line_wrap (GtkLabel * label, gboolean wrap);  
void gtk_label_set_text (GtkLabel * label, const gchar * str);
```



Button

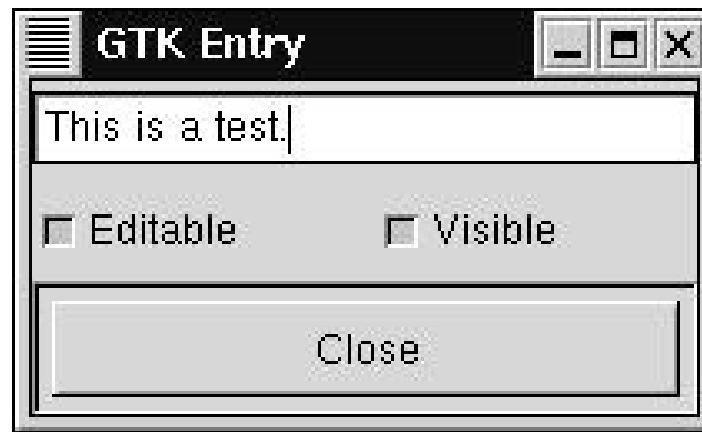
```
GtkWidget* gtk_button_new (void);
```

```
GtkWidget* gtk_button_new_with_label (const gchar *label);
```



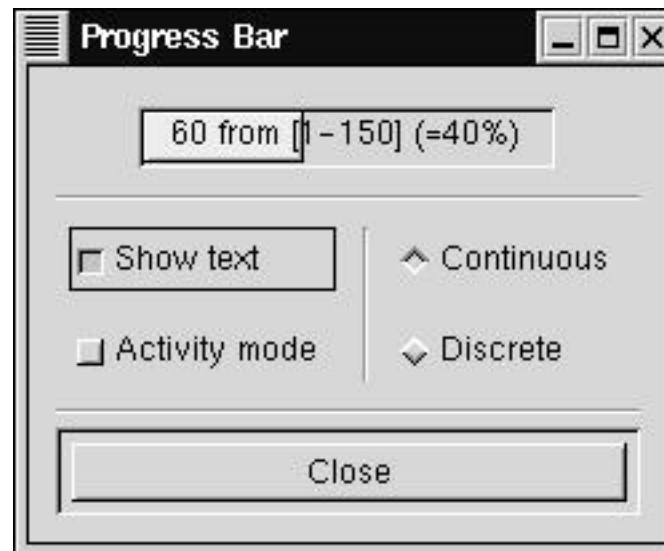
Entry

```
GtkWidget* gtk_entry_new (void);  
GtkWidget* gtk_entry_new_with_max_length (guint16 max);  
void gtk_entry_set_text (GtkEntry *entry, const gchar *text);  
gchar* gtk_entry_get_text (GtkEntry *entry);
```



Progressbar

```
GtkWidget* gtk_progress_bar_new (void);  
void ggtk_progress_bar_pulse (GtkProgressBar *pbar);
```



Menubar

```
GtkWidget* gtk_menu_bar_new (void);  
void gtk_menu_bar_append (GtkMenuBar *menu_bar,  
                          GtkWidget *child);  
void gtk_menu_bar_prepend (GtkMenuBar *menu_bar,  
                           GtkWidget *child);  
void gtk_menu_bar_insert (GtkMenuBar *menu_bar,  
                          GtkWidget *child, gint position);
```



Other Widgets (cont'd.)

**** Just remember to add your widgets to a container, and don't forget to show them. It would help if you show your container last, that way, all your widgets will be shown at the same time, instead of one after the other.**

Basic Layout

Fixed Container

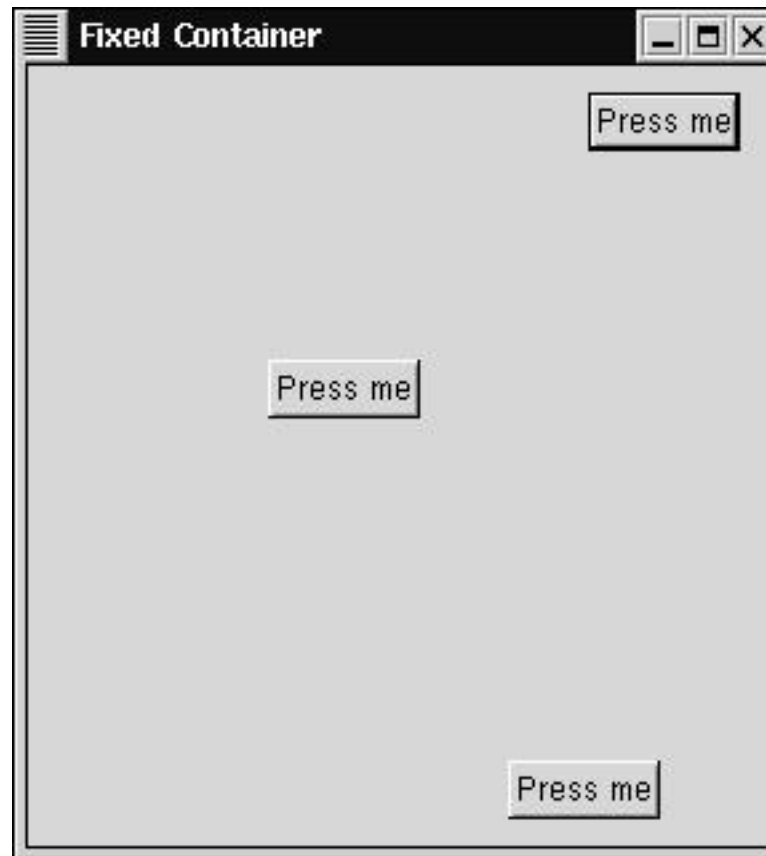
- ▶ Allows you to place widgets at a fixed position within its window, relative to its upper left hand corner.
- ▶ The position of the widgets can be changed dynamically.
- ▶ 3 functions associated with the fixed widget.

Fixed Container (cont'd.)

- ▶ *gtk_fixed_new* - to create a new Fixed Container
 - `GtkWidget* gtk_fixed_new(void);`
- ▶ *gtk_fixed_put* - to place widget in the container fixed at the position specified by x and y
 - `void gtk_fixed_put(GtkFixed *fixed, GtkWidget *widget, gint16 x, gint16 y);`
- ▶ *gtk_fixed_move* - allows the specified widget to be moved to a new position
 - `void gtk_fixed_move(GtkFixed *fixed, GtkWidget *widget, gint16 x, gint16 y);`

Fixed Container (cont'd.)

- ▶ Screenshot example



Layout Container

- ▶ Implements an infinite scrolling area.
- ▶ Have smooth scrolling even when you have many child widgets in the scrolling area.
- ▶ A layout container is created using:
 - `GtkWidget *gtk_layout_new(GtkAdjustment *hadjustment, GtkAdjustment *vadjustment);`

Layout Container (cont'd.)

► Add and move widgets:

- `void gtk_layout_put(GtkLayout *layout, GtkWidget *widget, gint x, gint y);`
- `void gtk_layout_move(GtkLayout *layout, GtkWidget *widget, gint x, gint y);`

► Set size of container:

- `void gtk_layout_set_size(GtkLayout *layout, guint width, guint height);`

Layout Container (cont'd.)

- ▶ Layout containers are one of the very few widgets in the GTK widget set that actively repaint themselves on screen as they are changed using the above functions (the vast majority of widgets queue requests which are then processed when control returns to the `gtk_main()` function).

Layout Container (cont'd.)

- ▶ Disable and enable repainting functionality:
 - `void gtk_layout_freeze(GtkLayout *layout);`
 - `void gtk_layout_thaw(GtkLayout *layout);`
- ▶ Manipulating the adjustment widgets:
 - `GtkAdjustment* gtk_layout_get_hadjustment(GtkLayout *layout);`
 - `GtkAdjustment* gtk_layout_get_vadjustment(GtkLayout *layout);`
 - `void gtk_layout_set_hadjustment(GtkLayout *layout, GtkAdjustment *adjustment);`
 - `void gtk_layout_set_vadjustment(GtkLayout *layout, GtkAdjustment *adjustment);`

Frames

- ▶ Used to enclose one or a group of widgets with a box which can optionally be labeled.

- ▶ To create a Frame:

```
GtkWidget *gtk_frame_new( const gchar *label );
```

- ▶ To change the text of the label:

```
void gtk_frame_set_label( GtkFrame *frame, const gchar *label );
```

Frames (cont'd.)

- ▶ To change the position of the label:

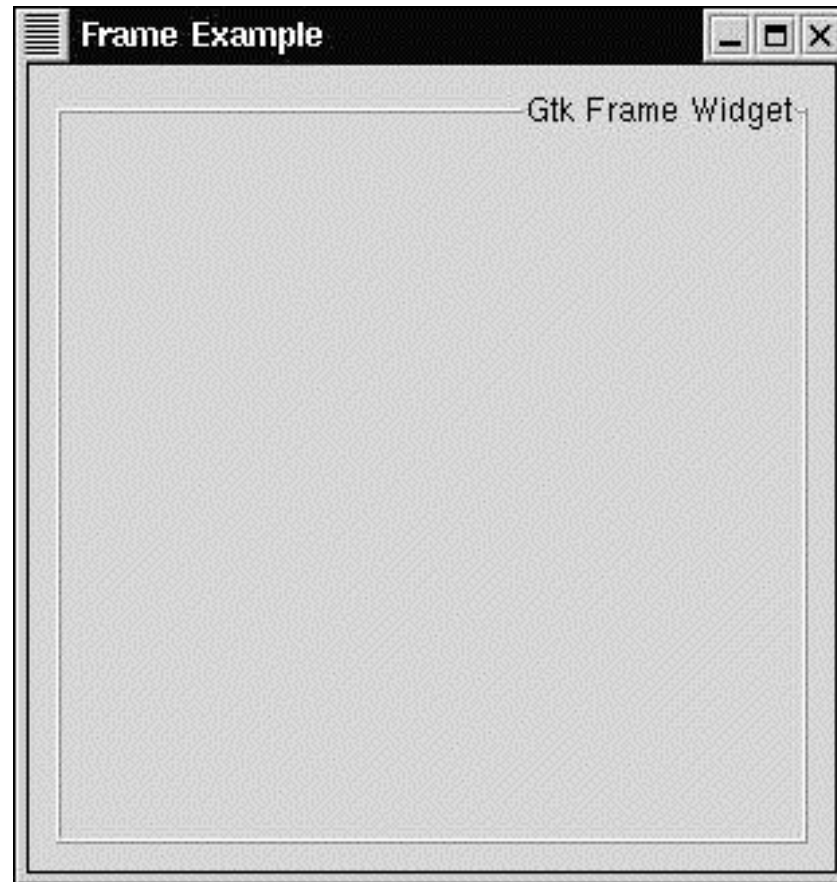
```
void gtk_frame_set_label_align(   GtkWidget *frame,  
                                gfloat      xalign,  
                                gfloat      yalign );
```

- ▶ To alter the style of the box that is used to outline the frame:

```
void gtk_frame_set_shadow_type( GtkWidget *frame,  
                               GtkShadowType type);
```

Frames (cont'd.)

- ▶ Screenshot example



Paned Window Widgets

- ▶ Used to divide an area into two parts, with the relative size of the two parts controlled by the user.
- ▶ The division can either be horizontal (HPaned) or vertical (VPaned).
- ▶ To create a new paned window:

```
GtkWidget *gtk_hpaned_new (void);
```

```
GtkWidget *gtk_vpaned_new (void);
```

Paned Window Widgets (cont'd.)

- ▶ To add child widgets to its two halves:

```
void gtk_paned_add1 (GtkPaned *paned, GtkWidget *child);
```

```
void gtk_paned_add2 (GtkPaned *paned, GtkWidget *child);
```

Paned Window Widgets (cont'd.)

- ▶ Screenshot example



Handling Events

Signals and Callbacks

- ▶ GTK is an event driven toolkit
- ▶ The passing of controls is done using the idea of “signals”
- ▶ When an event occurs, the appropriate signal will be “emitted” by the widget that was pressed.
- ▶ There are signals that all widgets inherit and there are signals that are widget specific

Signals and Callbacks (cont'd.)

► Examples

- `void GtkWidget::destroy (GtkWidget *, gpointer);`
- `void GtkWidget::show (GtkWidget *, gpointer);`
- `void GtkWidget::hide (GtkWidget *, gpointer);`
- `void GtkWidget::add (GtkWidget *, GtkWidget *, gpointer);`
- `void GtkWidget::changed (GtkWidget *, gpointer);`
- `void GtkWidget::orientation-changed (GtkWidget *, gint, gpointer);`
- `void GtkWidget::pressed (GtkWidget *, gpointer);`
- `void GtkWidget::released (GtkWidget *, gpointer);`
- `void GtkWidget::clicked (GtkWidget *, gpointer);`
- `void GtkWidget::toggled (GtkWidget *, gpointer);`
- `void GtkWidget::activate (GtkWidget *, gpointer);`
- `void GtkWidget::changed (GtkWidget *, gpointer);`

Signals and Callbacks (cont'd.)

► Signal handler:

- *gint gtk_signal_connect(GtkWidget *object, gchar *name, GtkSignalFunc func, gpointer func_data);*
 - 1st argument - the widget which will be emitting the signal
 - 2nd argument - the name of the signal you wish to catch
 - 3rd argument - the function you wish to be called when it is caught
 - 4th argument - the data you wish to have passed to this function

► “Callback function”

- *void callback_func(GtkWidget *widget, gpointer callback_data);*
 - 1st argument - a pointer to the widget that emitted the signal
 - 2nd argument - a pointer to the data given as the last argument to the `gtk_signal_connect()` function as shown above

Signals and Callbacks (cont'd.)

- ▶ Another call used is:

- `gint gtk_signal_connect_object(GtkWidget *object, gchar *name, GtkSignalFunc func, GtkWidget *slot_object);`
 - uses only one argument, a pointer to a GTK object

- ▶ The callback should be:

- `void callback_func(GtkWidget *object);`
 - the object is usually a widget

- ▶ The purpose of having 2 functions to connect signals is simply to allow the callbacks to have a different number of arguments.

Events

► There is a set of *events* that reflect the X event mechanism.

► These events are:

- event
- button_release_event
- motion_notify_event
- delete_event
- destroy_event
- expose_event
- key_press_event
- key_release_event
- enter_notify_event
- leave_notify_event
- configure_event
- focus_in_event
- focus_out_event
- map_event
- unmap_event
- property_notify_event
- selection_clear_event
- selection_request_event
- selection_notify_event
- proximity_in_event
- proximity_out_event
- drag_begin_event
- drag_request_event
- drag_end_event
- drag_enter_event
- drop_leave_event
- drop_data_available_event
- other_event

Events (cont'd.)

- ▶ Use `gtk_signal_connect` to connect a callback function to one of the events , using one of the event names as the name parameter.
- ▶ Callback function:
 - `gint callback_func(GtkWidget *widget, GdkEvent *event, gpointer callback_data);`

Events (cont'd.)

- ▶ GdkEvent is a C union structure whose type will depend upon which event has occurred
- ▶ In order for us to tell which event has been issued each of the possible alternatives has a type member that reflects the event being issued. The other components of the event structure will depend upon the type of the event.

Events (cont'd.)

► Possible values for the type:

- *GDK_NOTHING*
- *GDK_DESTROY*
- *GDK_MOTION_NOTIFY*
- *GDK_2BUTTON_PRESS*
- *GDK_BUTTON_RELEASE*
- *GDK_KEY_RELEASE*
- *GDK_LEAVE_NOTIFY*
- *GDK_CONFIGURE*
- *GDK_UNMAP*
- *GDK_SELECTION_CLEAR*
- *GDK_SELECTION_NOTIFY*
- *GDK_PROXIMITY_OUT*
- *GDK_DRAG_REQUEST*
- *GDK_DROP_LEAVE*
- *GDK_CLIENT_EVENT*
- *GDK_NO_EXPOSE*
- *GDK_DELETE*
- *GDK_EXPOSE*
- *GDK_BUTTON_PRESS*
- *GDK_3BUTTON_PRESS*
- *GDK_KEY_PRESS*
- *GDK_ENTER_NOTIFY*
- *GDK_FOCUS_CHANGE*
- *GDK_MAP*
- *GDK_PROPERTY_NOTIFY*
- *GDK_SELECTION_REQUEST*
- *GDK_PROXIMITY_IN*
- *GDK_DRAG_BEGIN*
- *GDK_DROP_ENTER*
- *GDK_DROP_DATA_AVAIL*
- *GDK_VISIBILITY_NOTIFY*
- *GDK_OTHER_EVENT*

Events (cont'd.)

► Example:

- To connect a callback function to one of the events
 - `gtk_signal_connect(GTK_OBJECT(button),
"button_press_event",
GTK_SIGNAL_FUNC(button_press_callback), NULL);`
- This assumes that `button` is a button widget.
- When the mouse is over the button and a mouse button is pressed, the function `button_press_callback` will be called.
 - `static gint button_press_callback(GtkWidget *widget,
GdkEventButton *event, gpointer data);`

Events (cont'd.)

- Note that we can declare the second argument as type `GdkEventButton` as we know what type of event will occur for this function to be called.
- The value returned from this function indicates whether the event should be propagated further by the GTK event handling mechanism. Returning
 - `TRUE` - indicates that the event has been handled, and that it should not propagate further.
 - `FALSE` - continues the normal event handling.

Hello World

```
#include <gtk/gtk.h>
void hello( GtkWidget *widget,gpointer  data )
{
    g_print ("Hello World\n");}
gint delete_event( GtkWidget *widget,GdkEvent *event,gpointer  data )
{
    /* If you return FALSE in the "delete_event" signal handler,
     * GTK will emit the "destroy" signal. Returning TRUE means
     * you don't want the window to be destroyed.
     * This is useful for popping up 'are you sure you want to quit?'
     * type dialogs. */
    g_print ("delete event occurred\n");
    return(TRUE);
}
void destroy( GtkWidget *widget,gpointer  data )
{
    gtk_main_quit();
}
```



```
/* Here we connect the "destroy" event to a signal handler.
 * This event occurs when we call gtk_widget_destroy() on the
window,
 * or if we return FALSE in the "delete_event" callback. */
gtk_signal_connect (GTK_OBJECT (window), "destroy",
                    GTK_SIGNAL_FUNC (destroy), NULL);
button = gtk_button_new_with_label ("Hello World");
/* When the button receives the "clicked" signal, it will call the
 * function hello() passing it NULL as its argument. The hello()
 * function is defined above. */
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (hello), NULL);

/* This will cause the window to be destroyed by calling
 * gtk_widget_destroy(window) when "clicked". Again, the destroy
 * signal could come from here, or the window manager. */
gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                           GTK_SIGNAL_FUNC (gtk_widget_destroy),
                           GTK_OBJECT (window));

    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button); gtk_widget_show (window); gtk_main ();
    return(0);
}
```

CS 159.3 A : GROUP 2

Ma. Angela Arellano

Joshua Jeremy Gonzalez

Christian Niño Jacinto

Jose Antonio Mella

Yvette Lourdes Ong

Ana Janina Pangilinan