Layout Containers

• Positioning widgets within a window can be difficult to do right. Different users will have different screen resolutions, fonts and colors. We also want to allow the user to change the size of the window and have the widgets inside the window resize and reposition.

• Modern GUI toolkits provide layout containers to simplify and automate many of these chores.
Layout Containers

• The first example (example-2a.c) we will look at uses the GtkGrid container to layout several buttons. The GtkGrid container is one of the most flexible and powerful containers available.
#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 *grid;
    GtkWidget *button;

    /* create a new window, and set its title */
    window = gtk_application_window_new (app);
    gtk_window_set_title (GTK_WINDOW (window), "Window");
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);

    /* Here we construct the container that is going pack our buttons */
    grid = gtk_grid_new ();

    /* Pack the container in the window */
    gtk_container_add (GTK_CONTAINER (window), grid);

    //xxxx
button = gtk_button_new_with_label("Button 1");
g_signal_connect(button, "clicked", G_CALLBACK(print_hello), NULL);

/* Place the first button in the grid cell (0, 0), and make it fill
 * just 1 cell horizontally and vertically (ie no spanning) */
gtk_grid_attach(GTK_GRID(grid), button, 0, 0, 1, 1);

button = gtk_button_new_with_label("Button 2");
g_signal_connect(button, "clicked", G_CALLBACK(print_hello), NULL);

/* Place the second button in the grid cell (1, 0), and make it fill
 * just 1 cell horizontally and vertically (ie no spanning) */
gtk_grid_attach(GTK_GRID(grid), button, 1, 0, 1, 1);

button = gtk_button_new_with_label("Quit");
g_signal_connect_swapped(button, "clicked", G_CALLBACK(gtk_widget_destroy), window);
/* Place the Quit button in the grid cell (0, 1), and make it span 2 columns. */
gtk_grid_attach (GTK_GRID (grid), button, 0, 1, 2, 1);

/* Now that we are done packing our widgets, we show them all in one go, by calling
 * gtk_widget_show_all() on the window. This call recursively calls gtk_widget_show()
 * on all widgets that are contained in the window, directly or indirectly. */
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;
}
Layout Containers

- `gtk_container_add()` is used to add widgets to simple containers. Here it is used to add the grid to the window. More complicated containers should use `gtk_box_pack_start()` or `gtk_grid_attach()` instead.

- A grid is created with `gtk_grid_new()`. Note that it is not necessary to specify the dimensions (rows and columns) of the grid.
Buttons are added to the grid using `gtk_grid_attach()`. The first two arguments are the grid and button widgets. The next four arguments are `left`, `top`, `width`, and `height`. The `left` (column) and `top` (row) coordinates are zero-indexed. The `width` and `height` are the number of columns and rows that the button should span.
Layout Containers

- Grids are the recommended layout containers, but boxes are also available.
- There are horizontal and vertical boxes. Widgets are packed horizontally into a horizontal box (hbox) and vertically into a vertical box (vbox). Boxes can be (and usually are) packed into other boxes to create different layouts.
Layout Containers

hbox

vbox

vbox

hbox
The next example (example-2b.c) is similar to the previous example but uses boxes.

`gtk_box_new()` is used to create layout containers (boxes):

```c
GtkWidget *gtk_box_new
  (GtkOrientation orient, gint spacing);
```

- `orient` is `GTK_ORIENTATION_VERTICAL` or `GTK_ORIENTATION_HORIZONTAL`. `spacing` is the spacing (in pixels) between elements.
Objects can be added into boxes with `gtk_box_pack_start()`.

This will add objects starting at the left (hbox) or top (vbox).

`gtk_box_pack_end()` will add objects starting at the right or bottom side.

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

- **box** is the box to be packed and **child** is the object we are packing into the box.

- If **expand** is TRUE the **child** is given extra space allocated to the **box**. The extra space is split among all objects with **expand** TRUE. **fill** determines whether the objects or the space around them expand.

- **padding** is extra space (in pixels) to put around the object (above the global amount (**spacing**) specified in the box struct).
Layout Containers

• For more advanced layout, GTK provides tables and notebooks.

• Tables allow widgets to be laid out in a 2D grid. Single widgets can span multiple rows or columns of the grid.

• Notebooks are tabbed window display objects.

• Refer to the “Layout Containers” section of the reference manual for further information.
• The general steps in creating a widget are:
  • Use one of the `gtk_*_new()` functions to create a new widget.
  • Connect signal handlers to the appropriate events.
  • Set the widget attributes.
  • Pack the widget into a container.
  • Display the widget using `gtk_widget_show()`.
Casting

- All widgets are derived from the **GObject** base class. Many of the GTK functions require that a derived object be cast to/from a base object of the appropriate type. Macros are provided to do this:

  `G_OBJECT(gobj), GTK_OBJECT(obj), GTK_WIDGET(wdgt), GTK_WINDOW(win), GTK_BOX(box), GTK_SIGNAL_FUNC(fnctn), GTK_BUTTON(bttn), GTK_CONTAINER(cntnr)`
Window Widgets

- Common window objects are: `GtkWindow`, `GtkDialog`, `GtkMessageDialog`

- A `GtkWindow` is a toplevel widget that can contain other widgets. It is created with a call to `gtk_window_new(type)` where the `type` can be `GTK_WINDOW_TOPLEVEL` or `GTK_WINDOW_POPUP` with windows of type `GTK_WINDOW_TOPLEVEL` being the most common.
Window Widgets

• A **GtkDialog** window is a convenient way to prompt the user for a small amount of input via a pop-up window.

• A **GtkMessageDialog** window is a simple Dialog window that is used to display error, warning and informational messages or ask simple yes/no type questions.
Display Widgets

• The most common display widgets are: **GtkLabel**, **GtkImage**, **GtkProgressBar**, and **GtkStatusBar**.

• **GtkLabel** widgets are useful for displaying small amounts of text. They are often used to label another widget such as a button. To create a label call `gtk_label_new(str)`. The label text may be changed with a call to `gtk_label_set_text()`.
A **GtkImage** widget is used to display an image. The `gtk_image_new_from_file()` routine is usually used to load a new image. Several common image formats are supported. You can also create an empty image with `gtk_image_new()` and then load the image later.
• The standard **GtkButton** creates a signal which clicked on.

• A **GtkToggleButton** remains “pressed-in” when clicked. It returns to normal when pressed again. It emits the **toggled** signal. A **GtkCheckButton** is similar but contains a check box next to the button label.

• A **GtkRadioButton** allows a single choice from a group of options.
Entry Widgets

• There are several widgets available for numeric and/or text entry. A **GtkEntry** is used for a single line of text. **GtkHScale** and **GtkVScale** are sliders for selecting a value from a range. A **GtkSpinButton** includes up and down increment buttons to allow the user to increase or decrease a value in the text display.
Other Widgets

- There are several other widgets in the toolkit. See the reference manual for the complete list.
- The **gtk3-demo** program is part of the gtk-3-examples package and demonstrates many of the widgets. Source code is provided.
- Also see the Widget Gallery section of the GTK+ 3 Reference Manual.
In-class Exercise

• The `layout.cpp` program provides a simple example of using boxes for layout.

• Modify `layout.cpp` to include a “Quit” button at the bottom of the window. Try adding the button into an hbox and adding that into the provided vbox. Compare to adding the button directly into the vbox.