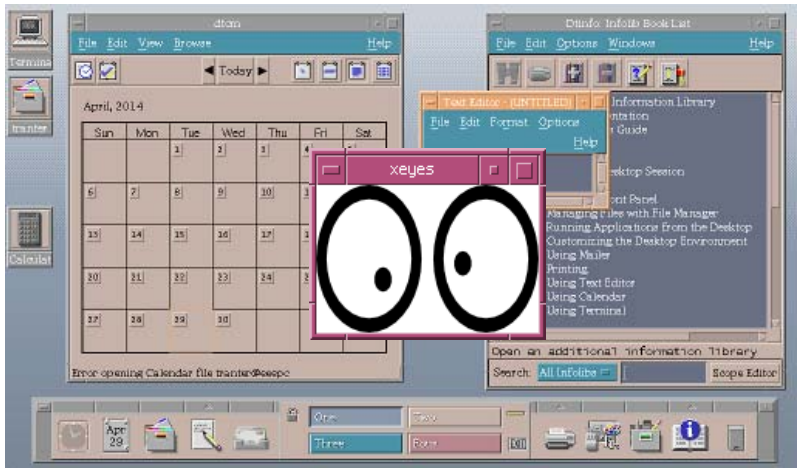
# CSE216 Programming Abstractions
## X Window Programming

YoungMin Kwon

# X Window System

- X Window System
  - Called X, X11, or X Windows
  - Network-based, Client-server user interface
  - Developed by MIT X consortium in mid 80s.



# X Window System

- Install GTK (GIMP Toolkit)
  - We will use GTK+ 3.0 in the class
  - Install GTK and other development environments

```
$ sudo apt-get install gnome-devel
```

  - GNOME environment uses GTK+ as a base
  - GTK+ uses the C programming language

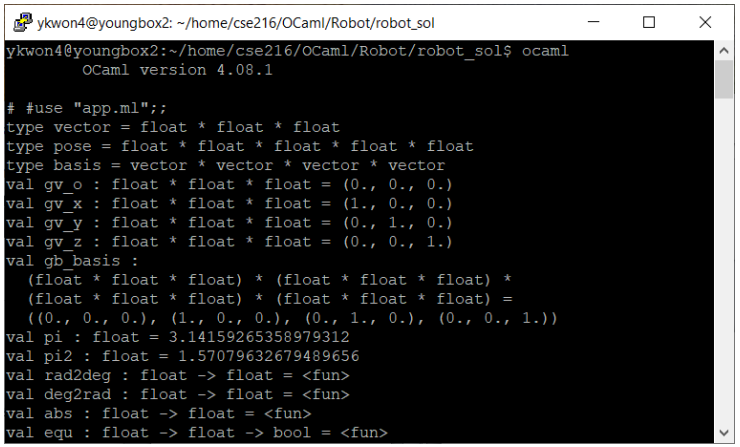
GIMP 2.2

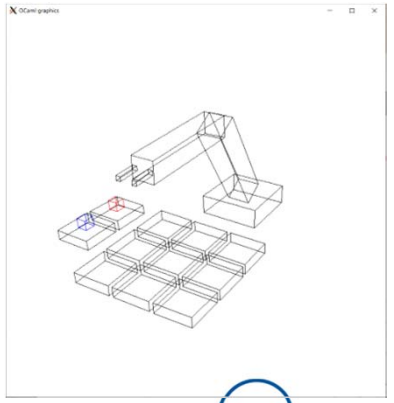
# Architecture of X

- X Server
  - Program that manages the screen and its input devices (keyboard, mouse)
    - Example: Xming, XQuartz, Xorg
  - Using X protocol, clients make request and receives events
    - Requests: create a window, draw a line, …
    - Events: key or mouse click, window resize, …

# Architecture of X

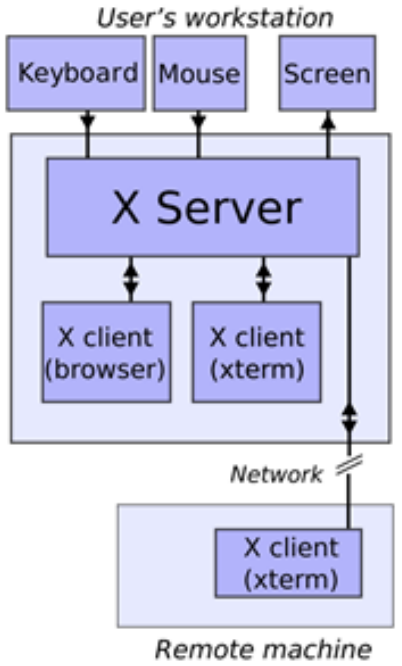
- X Client (Xlib)
  - Programs that make requests and handle events to/from X server
    - Example: OCaml running on AWS
  - Almost all client applications make calls to a library of functions known as Xlib



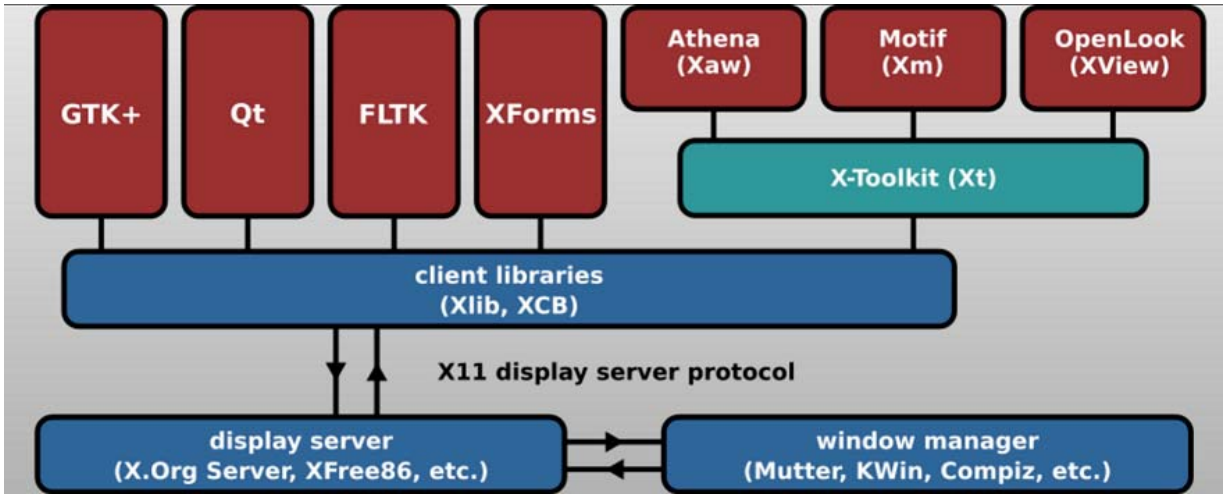
# Architecture of X

- X Protocol
  - Protocol between X server and X client
  - TCP/IP based
    - DISPLAY variable tells which X server to connect to
    - DISPLAY = 10.20.30.40:5.0 means screen 0 of X server 5 running at 10.20.30.40.



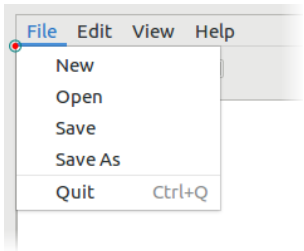
# Architecture of X

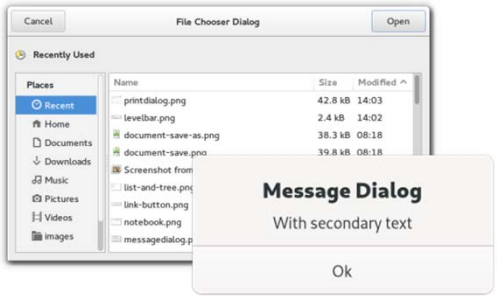
- Toolkits
  - An abstraction layer above Xlib
    - Abstracts the X Protocol
    - Provides a consistent look and feel
      - Look: how an application appears on the screen
      - Feel: how the user interacts with it

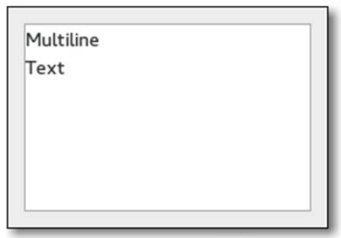


# Architecture of X

- Toolkits
  - Widgets
    - Source code that implements a user interface abstraction
    - Gadgets with their own X Window
      - Self-sufficient: repaint in response to events
    - Examples
      - GtkLabel, GtkButton, GtkMenu, GtkDialog, GtkFileChooserDialog, GtkText

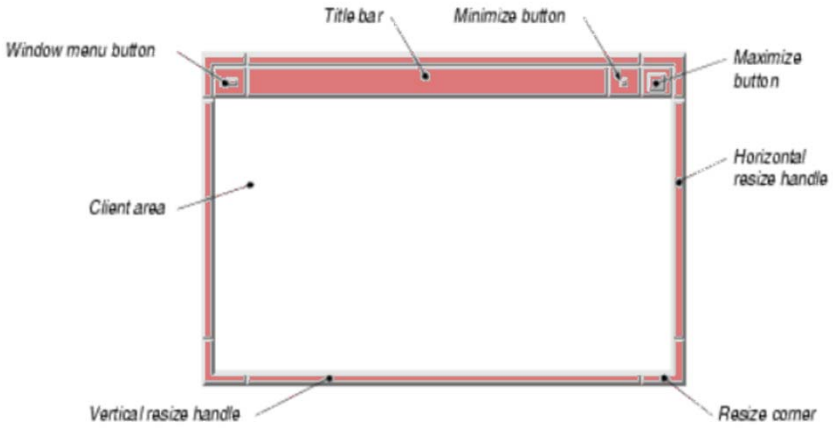



# Architecture of X

- Window Manager
  - A system software that controls the placement and appearance of windows
    - Controls to move, resize, minimize, delete windows…
  - Shell
    - Every visible top-level window has a widget called shell
    - Shells communicate with Window Managers




Motif window manager decorations

# First.c

- First.c
  - Create widgets

```
GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
GtkWidget *label  = gtk_label_new("Hello World!");
```

    - Toplevel windows can have Window Manager decorations

  - Add a widget to a container widget

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

  - Start the event loop

```
gtk_main();
```

```c
//first.c
#include <gtk/gtk.h>
int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);        //initialize libraries,
                                   //set up default signal handlers,
                                   //handle arguments

    GtkWidget *window =
        gtk_window_new(            //create a new window
            GTK_WINDOW_TOPLEVEL); //toplevel window has windows manager
                                   //decorations

    GtkWidget *label =
        gtk_label_new("Hello World!");  //create a new label

    gtk_container_add(             //add label to window
        GTK_CONTAINER(window),     //cast window to GtkContainer
        label);                    //toplevel window can have one child

    gtk_widget_show_all(window);  //display all widgets

    gtk_main();                    //start the event loop

    return 0;
}
```

# Setup and Compile

- To compile first.c

```
$ gcc first.c –o first `pkg-config --cflags --libs gtk+-3.0`
;; or download Makefile and run make first

$ ls
first.c first
```

# Execute

- To run
  - Run X server (xming, xquartz) from your computer
  - Run the client program (first) from AWS

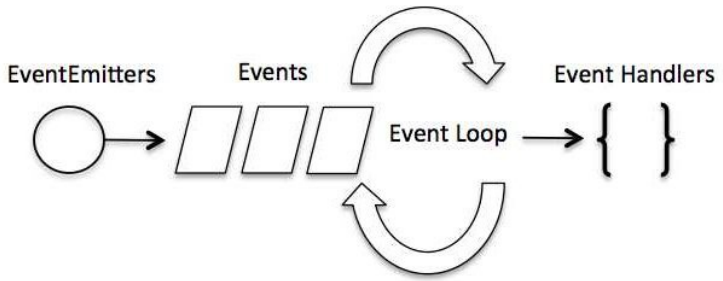
```
$ ./first
```

  - If you see this, congratulations on making your first X Window application


- To terminate
  - Ctrl + C (x button does not work and we will fix it!)

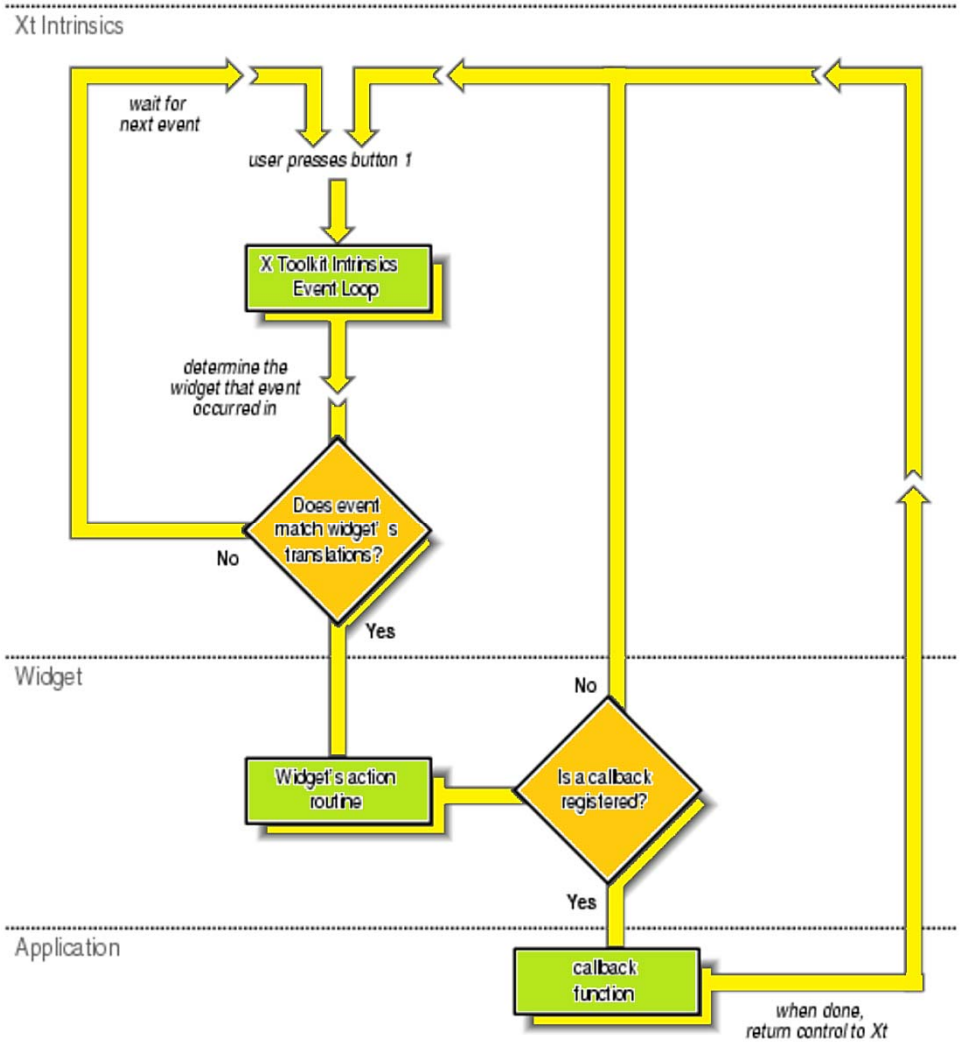
# Event Driven Programming

- Event driven programming
  - You do not call me, we will call you
  - Applications register callback functions (signal handlers)
  - Event loop will call callbacks later



# Event Driven Programming

- Xt/Motif Event Loop



# Hello.c

- Hello.c
  - Register a delete_event signal handler

```
g_signal_connect(           //add signal handler
    G_OBJECT(window),       //  instance
    "delete_event",         //  detailed_signal
    G_CALLBACK(on_delete),  //  c_handler
    NULL);                  //  data
```

    - register c_handler (on_delete) to instance (window) for detailed_signal (delete_event).
    - data (NULL) is a parameter to c_handler

  - To exit the event loop

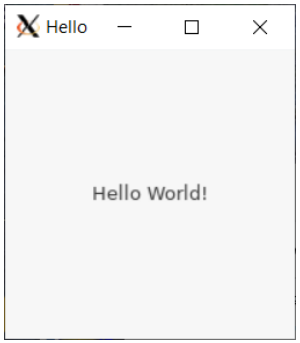
```
gtk_main_quit();
```

```c
//hello.c

#include <stdio.h>
#include <gtk/gtk.h>
void on_delete(          //callback (signal handler) for delete event
    GtkWidget *widget,   //the widget that made the event
    gpointer data)       //client data
{
    printf("bye.\n");
    gtk_main_quit();     //exit the event loop
}

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

    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title(                //set title
        GTK_WINDOW(window), "Hello");    //  cast window to GtkWindow

    gtk_widget_set_size_request(         //set window size
        GTK_WIDGET(window), 200, 200);   //  cast window to GtkWidget
…
```

```
    g_signal_connect(                       //add signal handler
        G_OBJECT(window),                   //  instance
        "delete_event",                     //  detailed_signal
        G_CALLBACK(on_delete),              //  c_handler
        NULL);                              //  data

    GtkWidget *label = gtk_label_new("Hello World!");
    gtk_container_add(GTK_CONTAINER(window), label);

    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

To compile and execute

```
$ make hello

$ ./hello
```



# Hello.c

- Makefile

```
INCLUDES = $(shell pkg-config --cflags gtk+-3.0)
LIBS     = $(shell pkg-config --libs gtk+-3.0)

RM   = rm
TRUE = true

.SUFFIXES:
.SUFFIXES: .c .o
.c.o:; gcc $(INCLUDES) -c $< -o $@

first: first.o
        gcc -o $@ $< $(LIBS)

hello: hello.o
        gcc -o $@ $< $(LIBS)

clean:
        $(RM) *.o | $(TRUE)
```

# Button.c

- Tasks
  - Add a button to the toplevel window
  - Add a click handler
  - Pass a client data to the click handler

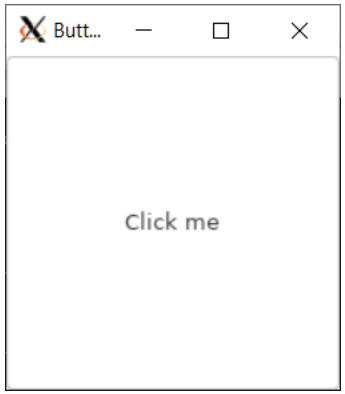
```c
//button.c

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

void on_click(GtkWidget *widget, gpointer data) {
    printf("data: %s\n", (char*)data);
}

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), "Button");
    gtk_widget_set_size_request(GTK_WIDGET(window), 200, 200);

    g_signal_connect(G_OBJECT(window), "delete_event",
        G_CALLBACK(on_delete), NULL);
```

```
    GtkWidget *button = gtk_button_new_with_label("Click me");

    g_signal_connect(           //add a callback to button
        G_OBJECT(button),       //  the target instance as an object
        "clicked",              //  clicked event
        G_CALLBACK(on_click),   //  add on_click as a callback
        "I am clicked");        //  parameter (data) to on_click

    gtk_container_add(GTK_CONTAINER(window), button);


    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

To compile and execute

- Add lines for button to Makefile

```
$ make button
$ ./button
```



# Container

- Container Widget
  - A widget that manages other widgets
  - Visibility: many containers are invisible, but they manage the geometry of their children
  - Layout policy: how does the container place its children within the container
    - Children added from the left edge or the right edge of the container
    - Children anchored to certain edge of the container

# GtkContainer

- GtkContainer
  - Parent class of a broad category of widgets for managing layouts
  - Some of its functions

```
//add widget to container
void gtk_container_add(GtkContainer *container, GtkWidget*widget);

//remove widget from container
void gtk_container_remove(GtkContainer *container, GtkWidget *widget);

//invoke callback for each child of container
void gtk_container_foreach(
        GtkContainer *container,
        GtkCallback callback,
        gpointer callback_data);
```

# GtkBox

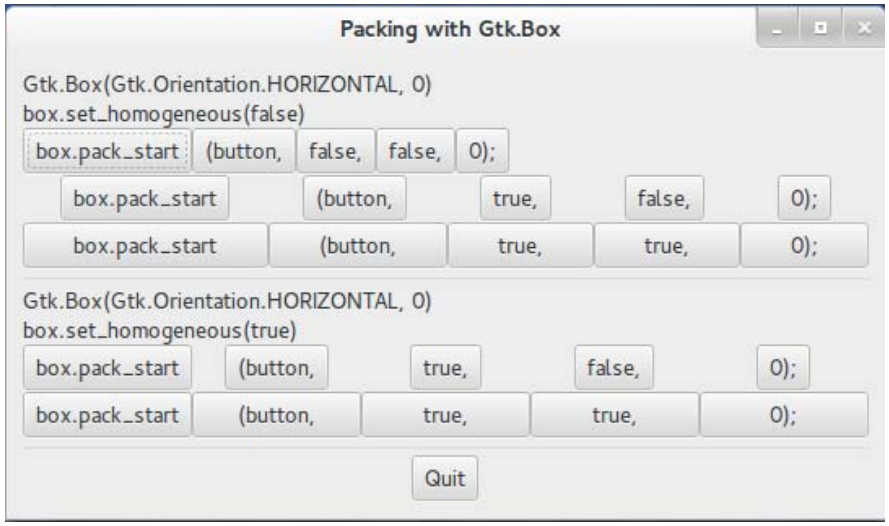
- GtkBox
  - One of the most popular containers
    - Almost all layouts can be achieved using GtkBox
  - Some attributes
    - Homogeneous: if true all children have the same size
    - Spacing: space between child widgets
    - Expand: (homogeneous is false) if true, child widgets are packed so that the entire area of box is used
    - Fill: (when expand is true) if true, draw to fill the space given to them; if false, draw large enough to draw its contents
    - Padding: add pixels around what would be normally allocated for child

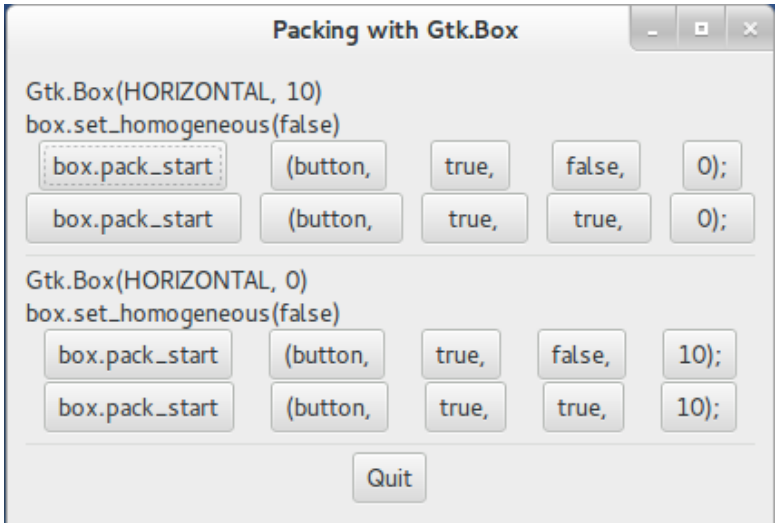
# GtkBox

- GtkBox
  - Some of its functions

```
typedef enum {
  GTK_ORIENTATION_HORIZONTAL,
  GTK_ORIENTATION_VERTICAL
} GtkOrientation;

GtkWidget* gtk_box_new(GtkOrientation orientation, gint spacing);

void gtk_box_set_homogeneous(GtkBox *box, gboolean homogeneous);
```

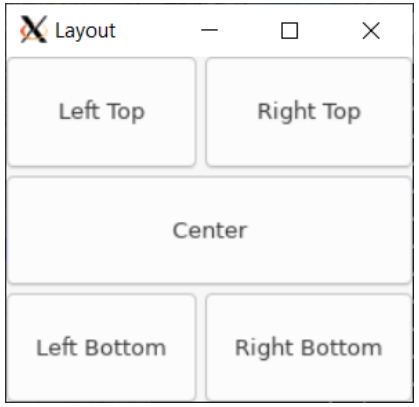
```
//pack a child from the left or from the top
void gtk_box_pack_start(GtkBox *box, GtkWidget *child,
        gboolean expand, gboolean fill, guint padding);

//pack a child from the right or from the bottom
void gtk_box_pack_end(GtkBox *box, GtkWidget *child,
        gboolean expand, gboolean fill, guint padding);
```



# Layout.c

- Tasks
  - Toplevel window can have only one child
  - Using Container (GtkBox), add multiple children to the application
  - Make a table of buttons using GtkBoxes (virtual and horizontal directions)



```c
//layout.c

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

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), "Layout");
    gtk_widget_set_size_request(GTK_WIDGET(window), 200, 200);
    g_signal_connect(G_OBJECT(window), "delete_event",
        G_CALLBACK(on_delete),NULL);

    GtkWidget *vbox = gtk_box_new(
        GTK_ORIENTATION_VERTICAL,   //orientation
        5);                         //spacing
    gtk_container_add(GTK_CONTAINER(window), vbox);
```

```
//top hbox
GtkWidget *hbox_top = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 5);
gtk_box_pack_start(
    GTK_BOX(vbox),  //box
    hbox_top,       //child
    TRUE,           //expand
    TRUE,           //fill
    0);             //padding

GtkWidget *btn_lt = gtk_button_new_with_label("Left Top");
g_signal_connect(G_OBJECT(btn_lt), "clicked",
    G_CALLBACK(on_click), "Left-top");
gtk_box_pack_start(GTK_BOX(hbox_top), btn_lt, TRUE, TRUE, 0);

GtkWidget *btn_rt = gtk_button_new_with_label("Right Top");
g_signal_connect(G_OBJECT(btn_rt), "clicked",
    G_CALLBACK(on_click), "Right-top");
gtk_box_pack_start(GTK_BOX(hbox_top), btn_rt, TRUE, TRUE, 0);

//center button
GtkWidget *btn_c = gtk_button_new_with_label("Center");
g_signal_connect(G_OBJECT(btn_c), "clicked",
    G_CALLBACK(on_click), "Center");
gtk_box_pack_start(GTK_BOX(vbox), btn_c, TRUE, TRUE, 0);
```

```
    //bottom hbox
    GtkWidget *hbox_bot = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 5);
    gtk_box_pack_start(GTK_BOX(vbox), hbox_bot, TRUE, TRUE, 0);

    GtkWidget *btn_lb = gtk_button_new_with_label("Left Bottom");
    g_signal_connect(G_OBJECT(btn_lb), "clicked",
        G_CALLBACK(on_click), "Left-bottom");
    gtk_box_pack_start(GTK_BOX(hbox_bot), btn_lb, TRUE, TRUE, 0);

    GtkWidget *btn_rb = gtk_button_new_with_label("Right Bottom");
    g_signal_connect(G_OBJECT(btn_rb), "clicked",
        G_CALLBACK(on_click), "Right-bottom");
    gtk_box_pack_start(GTK_BOX(hbox_bot), btn_rb, TRUE, TRUE, 0);

    gtk_widget_show_all(window);
    gtk_main();

    return 0;
}
```

To compile and execute

- Add lines for layout to Makefile

```
$ make layout
$ ./layout
```