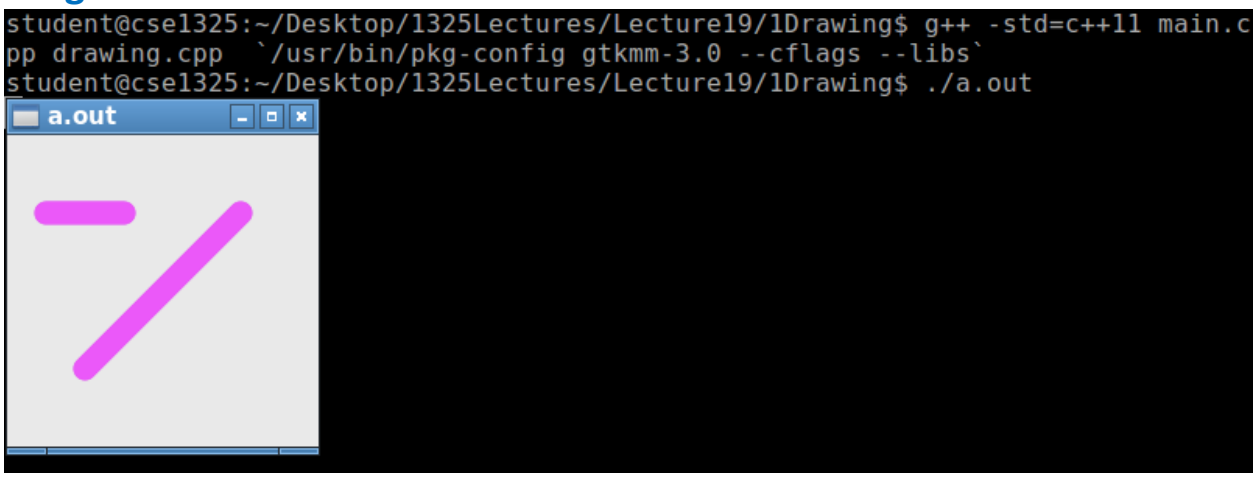
GUIs

Program 1:



drawing.h

```
#ifndef DRAW_H
#define DRAW_H

#include <iostream>
#include <gtkmm.h>

//DrawingArea is a widget.  It is a blank space that you can draw in
class Draw_example : public Gtk::DrawingArea
{
public:
 Draw_example();

protected:
 //this is the function that shows what we draw-it is an overridden function from the DrawingArea class
 //Cairo is a graphics library
 virtual bool on_draw(const Cairo::RefPtr<Cairo::Context>& cr);

};

#endif
```

drawing.cpp

```
#include "drawing.h"

Draw_example::Draw_example()
{
}

//https://www.cairographics.org/documentation/cairomm/reference/classCairo_1_1Context.html
```

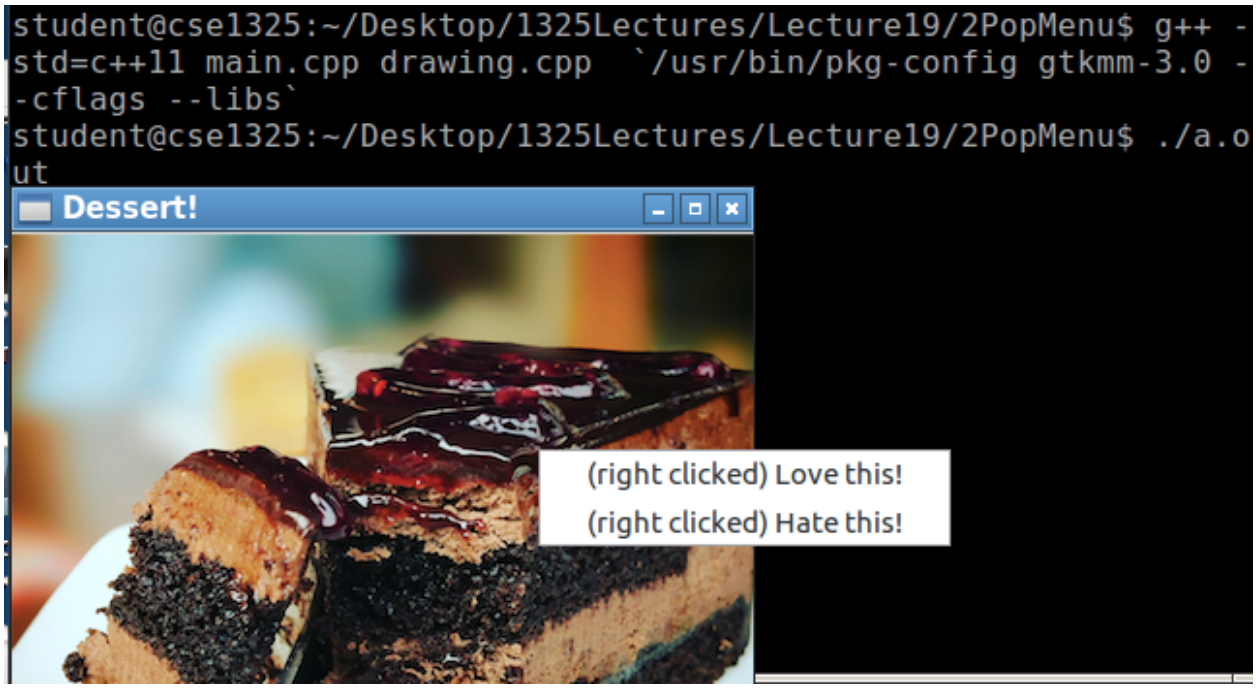
```
bool Draw_example::on_draw(const Cairo::RefPtr<Cairo::Context>& cr)
{
   //set how the line looks:
   cr->set_source_rgb(1.0, 0.0, 1.0);

   cr->set_line_width(15.0);

   cr->set_line_cap(Cairo::LINE_CAP_ROUND); //enums:
https://www.cairographics.org/documentation/cairomm/reference/namespaceCairo.html


   //draw the lines
   cr->move_to(150, 50);
   cr->line_to(50, 150);
   cr->move_to(75,50);
   cr->line_to(25, 50);

   cr->stroke();

   return true;
}
```

main.cpp

```
#include "drawing.h"
#include <gtkmm.h>


int main(int argc, char* argv[])
{
   Glib::RefPtr<Gtk::Application> app = Gtk::Application::create(argc, argv, "www.uta.edu");
   Gtk::Window window;
   //create a drawing
   Draw_example d;

   //add drawing to the window and make it visible
   window.add(d);
   d.show();

   return app->run(window);
}
```

Program 2:

We will see mouse event handling in this example (we have already dealt with signal handlers from buttons).



drawingarea.h

```
#ifndef DRAW_H
#define DRAW_H

#include <gtkmm.h>
#include <iostream>


class Drawing_example : public Gtk::DrawingArea
{
public:
 Drawing_example();

protected:

 virtual bool            on_draw(const Cairo::RefPtr<Cairo::Context>& cr);

 //When the mouse is clicked
 bool                    on_button_press_event(GdkEventButton *event);

 //When a popup item is clicked
 void                    event1();
 void                    event2();

private:

  //Gdk is a library-You can look all these additional libraries up to learn more about the classes we are using inside
 // Pixbuf is for image manipulation
 Glib::RefPtr<Gdk::Pixbuf>   image;
```

```
 Gtk::Menu              popup_menu;
 Gtk::MenuItem          menu1;
 Gtk::MenuItem          menu2;

};
#endif
```

drawing.cpp

```
#include "drawing.h"



Drawing_example::Drawing_example()
{
 image = Gdk::Pixbuf::create_from_file("yummy.png");

 // for mouse events (an event is when something occurs-like a mouse click)
 add_events(Gdk::BUTTON_PRESS_MASK);


 //the official website was down, so here is another place to get a class reference:
http://pascal.rigaud4.free.fr/Programmation/GTK/GTKMMDoc/GTKMM/www.gtkmm.org/docs/gtkmm-2.4/docs/reference/html/classGtk_1_1MenuItem.html
 //when the popup menu is up
 menu1.set_label("(right clicked) Love this!");
 menu1.signal_activate().connect(sigc::mem_fun(*this,&Drawing_example::event1));
 popup_menu.append(menu1);

 menu2.set_label("(right clicked) Hate this!");
 menu2.signal_activate().connect(sigc::mem_fun(*this,&Drawing_example::event2));
 popup_menu.append(menu2);

 popup_menu.show_all();

 //connect menu and widget
 popup_menu.accelerate(*this);
}


// Signal handlers
void Drawing_example::event1()
{
 std::cout << "Love this!" << std::endl;
}

void Drawing_example::event2()
{
 std::cout << "Hate this!" << std::endl;
}
```

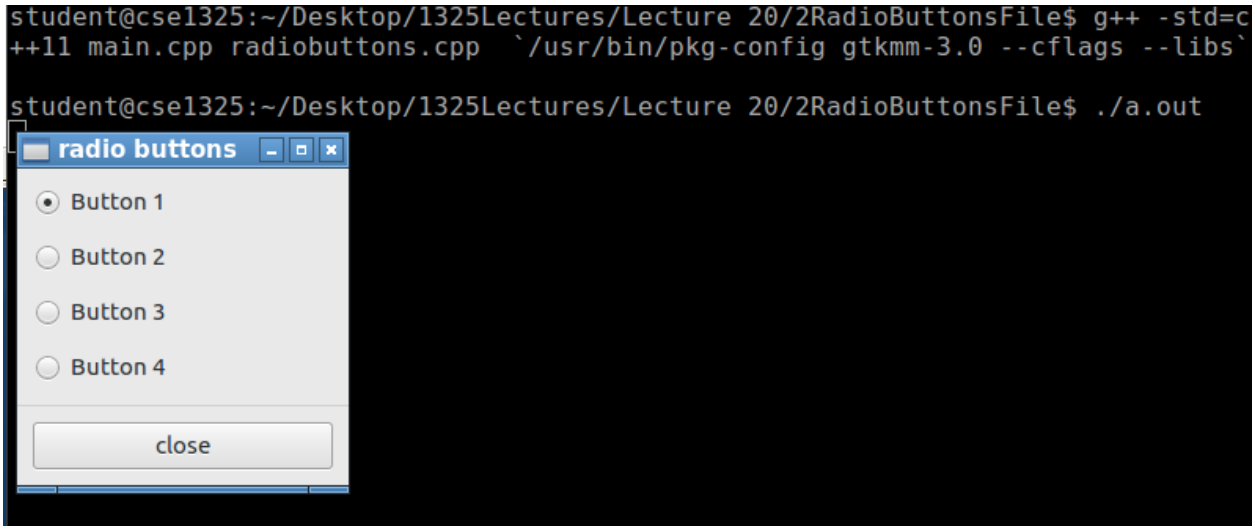
```
// Mouse button press event (event handler).  We are passing in the event (a mouse click)
//You can see a list of all events here:
https://developer.gnome.org/gdk3/stable/gdk3-Event-Structures.html#GdkEventButton

bool Drawing_example::on_button_press_event(GdkEventButton *event)
{
 //check if the event is a right click (which is how the popup menu should come up)
 if( (event->type == GDK_BUTTON_PRESS) && (event->button == 3) )
 {
 // Display the popup menu if it is a right button click
 m_Menu_Popup.popup(event->button, event->time);  //info about the meaning of the arguments:
https://developer.gnome.org/gdk3/stable/gdk3-Event-Structures.html#GdkEventButton

 return true;
 }

 return false;
}

bool Drawing_example::on_draw(const Cairo::RefPtr<Cairo::Context>& cr)
{
 Gdk::Cairo::set_source_pixbuf(cr, image, 0,0);
 cr->rectangle(0, 0, image->get_width(), image->get_height());
 cr->fill();
 return true;
}
```

main.cpp

```
#include "drawing.h"
#include <gtkmm.h>


int main(int argc, char* argv[])
{
 Glib::RefPtr<Gtk::Application> app = Gtk::Application::create(argc, argv, "www.uta.edu");
 Gtk::Window window;

  Drawing_example d;

 window.add(d);
 window.resize(400,300);
 window.set_title("Dessert!");
 d.show();

 return app->run(window);
}
```

Program 3:

Use a vector to provide the radio buttons:



radiobuttons.h

```
#ifndef RADIOBUTTONS_H
#define RADIOBUTTONS_H

#include <gtkmm.h>

class Radiobuttons : public Gtk::Window
{

std::vector<Gtk::RadioButton*> all_buttons;  //a vector of RadioButton widget pointers

public:
  Radiobuttons(std::vector<std::string> button_stuff);
  virtual ~Radiobuttons();
  void make_buttons(std::vector<std::string> buttons);

protected:
  //signal handlers:
  void on_button_clicked();

  // widgets:
  Gtk::Box box1, box2, box3;
  Gtk::Separator line;
  Gtk::Button close;
};

#endif //RADIOBUTTONS_H
```

radiobuttons.cpp

```
#include "radiobuttons.h"

//create RadioButtons (using a vector of strings with the name of each button) and put them in the vector we declared in the .h file (notice we are making pointers)
void Radiobuttons::make_buttons(std::vector<std::string> buttons)
{
  for(int i=0;i<buttons.size();i++)
  {
 Gtk::RadioButton* b=new Gtk::RadioButton(buttons[i]);
          all_buttons.push_back(b);
  }

}

//passing in the vector of strings for the radio buttons (used above)
Radiobuttons::Radiobuttons(std::vector<std::string> button_stuff) :
 box1(Gtk::ORIENTATION_VERTICAL),
 box2(Gtk::ORIENTATION_VERTICAL, 10),
 box3(Gtk::ORIENTATION_VERTICAL, 10),
 close("close")
{
 // set title and border of the window
 set_title("radio buttons");
 set_border_width(0);
 make_buttons(button_stuff); //calling function above

 //in order to group radio buttons together (so they function as a group) we call a function called join_group. Every button we made (and kept as a pointer in the vector) is being added to join group)
 for(int i=1;i<button_stuff.size();i++)
 {
    all_buttons[i]->join_group(*all_buttons[0]);
 }


 // add outer box to the window (because the window can only contain a single widget)
 add(box1);

 //put the inner boxes and the separator in the outer box:
 box1.pack_start(box2);
 box1.pack_start(line);
 box1.pack_start(box3);

 // set the inner boxes' borders
 box3.set_border_width(10);
 box2.set_border_width(10);

 // put the radio buttons in Box1:
 for (int i=0;i<button_stuff.size();i++)
 {
```

```
   box2.pack_start(*all_buttons[i]);
  }

  // put Close button in Box2:
  box3.pack_start(close);

  // Make the button the default widget
  close.set_can_default();
  close.grab_default();

  // connect the clicked signal of the button to Radiobuttons::on_button_clicked()
  close.signal_clicked().connect(sigc::mem_fun(*this, &Radiobuttons::on_button_clicked) );

  // show all children of the window
  show_all_children();
}

//you should delete pointers when you're done-I didn't in this example

Radiobuttons::~Radiobuttons()
{
}

void Radiobuttons::on_button_clicked()
{
  hide(); //close the application.
}
```

main.cpp

```
#include "radiobuttons.h"
#include <gtkmm.h>

int main(int argc, char *argv[])
{
   Gtk::Main app(argc, argv);

  //this will be on our radio buttons:
  std::vector<std::string> info={"Button 1", "Button 2", "Button 3", "Button 4"};
  Radiobuttons buttons(info);

  Gtk::Main::run(buttons);

  return 0;
}
```