Static:

Conceptually, *static* is used to represent situations where a characteristic or function is not a specific characteristic of a specific object created from a class. While programming, static simply means we can call a function or variable without creating an object.

For example, when we create Person objects from a Person class, each person object will have a specific name-that name is specific to the object created (this would not need static). See example below for when static would be suitable.

Example:

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
This employee's id: 0
This employee's id: 1
This employee's id: 2
0
1
2
Total employees:3
Total employees:3
Total employees:3
```

```
#include <iostream>

using namespace std;

class Employee{

public:
  static int total_employees;  //this represents all employees ever created from this class.  this value is not specific to any employee-instead it is specific to the class itself
  int employee_id; //I will assign a unique id to each employee-it will be the number of employees at that time. For example, if I am creating an Employee object and I have already created 2 before, then this new one will be the third so I will assign it 3

  Employee()
  {
    employee_id=total_employees; //setting a unique ID to each object made from this class
    cout<<"This employee's id: "<<employee_id<<endl;
    total_employees++; //we increment by one every time-this value will always be visible to any object created (and the class itself )
  }
};
```

```
int Employee::total_employees=0; //we start the value off at 0.


int main(int argc, char **argv)
{
  Employee e1;
  Employee e2;
  Employee e3;

  cout<<e1.employee_id<<endl;
  cout<<e2.employee_id<<endl;
  cout<<e3.employee_id<<endl;

  cout<<"Total employees:"<<e1.total_employees<<endl;
  cout<<"Total employees:"<<e2.total_employees<<endl;
  cout<<"Total employees:"<<Employee::total_employees<<endl;

  //cout<<Employee::employee_id<<endl; can't do this because employee_id is not static-it specific to a specific
object created
}
```

Example:

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
16
9
```

```
#include <iostream>
using namespace std;

class Little_math{

public:
  static int square(int num) //a static function means we can call it without using an object (but we still can
create an object to use it if we wanted to)
  {
    return num*num;
  }
};

int main(int argc, char **argv)
{
  int value=Little_math::square(4); //calling directly from the class (without an object)
  cout<<value<<endl;

  Little_math m1; //we could also do this
```

```
  int value2=m1.square(3);
  cout<<value2<<endl;
}
```

Friend classes:

A friend class can access private and protected members in another class (remember that normally non-members cannot access private or protected data-*data hiding*):

```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out
Flavor is: Butter
Price is: 7.99
```

Example 1:

```
#include <iostream>
#include <string>

class Popcorn{

  std::string flavor;

protected:
  float price;

public:
   /*Movie_goer ->accesses private Popcorn attributes*/
   friend class Movie_goer; /*Popcorn is now a friend class of Movie_goer so Movie_goer can access private and protected members.  Note that we would have to declare Movie_goer as a friend class to Popcorn to have it access private and protected members of Popcorn (see next example).  Just because class A is friends with class B does not mean class B is friends with class A-you would need to declare them both as friends.*/

   Popcorn(std::string flavor, float price)
   {
     this->flavor=flavor;
     this->price=price;
   }

};

class Movie_goer{
  std::string name;
  bool likes_popcorn;

public:
```

```
 Movie_goer(std::string name, bool likes_popcorn)
 {
  this->name=name;
  this->likes_popcorn=likes_popcorn;
 }

 void see_info(Popcorn p) /*we can access private and protected variables from Popcorn from a function in
Movie_goer since we declared Popcorn as a friend class of Movie_goer*/
 {
  std::cout<<"Flavor is: "<<p.flavor<<std::endl;
 std::cout<<"Price is: $"<<p.price<<std::endl;
 }
};


int main(int argc, char **argv)
{
      Movie_goer m1("Bobby", true);
      Popcorn p1("Butter", 7.99);

      m1.see_info(p1);
}
```

Example 2:

```
computer$ g++ -std=c++11 practice.cpp
practice.cpp:44:28: error: 'name' is a private member of 'Movie_goer'
        std::cout<<"Name is: "<<m.name<<std::endl;
                                  ^
practice.cpp:28:15:       implicitly declared private here
  std::string name;
              ^
practice.cpp:45:29: error: 'likes_popcorn' is a private member of 'Movie_goer'
  std::cout<<"Bool is: "<<m.likes_popcorn<<std::endl;//note 1 means true, 0 means
false
                             ^
practice.cpp:29:8:       implicitly declared private here
  bool likes_popcorn;
       ^
2 errors generated.
```

Notice here that Popcorn can't access private members of Movie_goer even though it's a friend (doesn't work both ways). We would have to declare them both as friends to allow access to both.

```
#include <iostream>
#include <string>

class Movie_goer; //forward declaration (so we can access Movie_goer object before as parameter in first class
even though we haven't made the class yet)

class Popcorn{

 std::string flavor;
```

```
protected:
  float price;

public:

   friend class Movie_goer; /*Popcorn is a friend class of Movie_goer so Movie_goer can access private and protected members, but it does not work the other way around (Popcorn cannot access Movie_goer info-see error above)*/

   Popcorn(std::string flavor, float price)
   {
     this->flavor=flavor;
     this->price=price;
   }

   void see_info(Movie_goer m); //function prototype-we have to define it after we declare the Movie_goer class (otherwise we will get an error)

};

class Movie_goer{
  std::string name;
  bool likes_popcorn;

public:

  Movie_goer(std::string name, bool likes_popcorn)
  {
    this->name=name;
    this->likes_popcorn=likes_popcorn;
  }


};

void Popcorn::see_info(Movie_goer m) //we are declaring the function after we declare the Movie_goer class (so we can use it)
{
  std::cout<<"Name is: "<<m.name<<std::endl;
  std::cout<<"Bool is: "<<m.likes_popcorn<<std::endl;//note 1 means true, 0 means false
}



int main(int argc, char **argv)
{
       Movie_goer m1("Bobby", true);
       Popcorn p1("Butter", 7.99);

  p1.see_info(m1);
```

}

Example 3:

By declaring both as friends, both can access anything private or protected in the other:

```
#include <iostream>
#include <string>

class Movie_goer;

class Popcorn{

  std::string flavor;

protected:
  float price;

public:

   friend class Movie_goer; //friend of Movie_goer

   Popcorn(std::string flavor, float price)
   {
     this->flavor=flavor;
     this->price=price;
   }

   void see_info(Movie_goer m);

};

class Movie_goer{
  std::string name;
  bool likes_popcorn;

public:

  Movie_goer(std::string name, bool likes_popcorn)
  {
    this->name=name;
    this->likes_popcorn=likes_popcorn;
  }

  friend class Popcorn; //friend of popcorn

  void see_info(Popcorn p) //we can access private and protected variables from Popcorn
  {
```

```
   std::cout<<"Flavor is: "<<p.flavor<<std::endl;
 std::cout<<"Price is: $"<<p.price<<std::endl;
  }
};

void Popcorn::see_info(Movie_goer m) //we can access private and protected variables from Movie_goer
{
 std::cout<<"Name is: "<<m.name<<std::endl;
 std::cout<<"Bool is: "<<m.likes_popcorn<<std::endl;//note 1 means true, 0 means false
}



int main(int argc, char **argv)
{
      Movie_goer m1("Bobby", true);
      Popcorn p1("Butter", 7.99);

 p1.see_info(m1);
}
```

You can also declare functions as friends:

Example 4:

```
#include <iostream>
using namespace std;

class Paper {
   int width, height;
  public:
   Paper()
   {
   }
   Paper (int x, int y) : width(x), height(y) //different syntax to give values
   {
   }
   int area()
   {
       return width * height;
   }
   friend Paper friendfunction (Paper& p);  //friendfunction is a friend of Paper, so it can access private members
(it is not actually part of the Paper class-notice when we define below we are not using the scope resolution
operator to say it "lives" in Paper class).
};

Paper friendfunction (Paper& p) //definition of friendfunction
{
```

```
  Paper r;
  r.width = p.width*5;
  r.height = p.height*5;
  return r;
}

int main (int argc, char **argv) {
  Paper foo;
  Paper p1 (2,3);
  foo = friendfunction (p1);
  cout << foo.area() << '\n';
  return 0;
}
```

GUIs:

Note: I am using the virtual machine for all these examples.

Note:

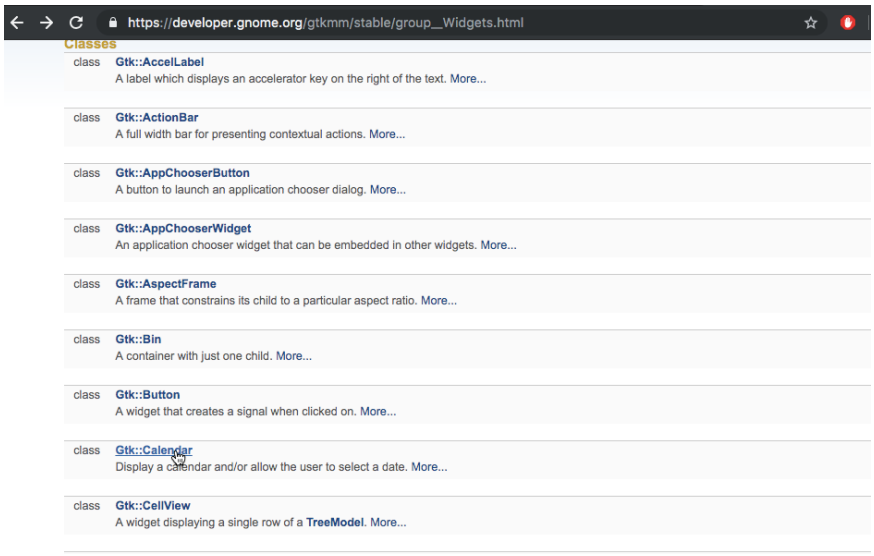
Obviously, I can't show you guys EVERYTHING-I can only do a few examples using GUIs. My goal is to show you a few examples showing the possibilities of GUIs using GTKMM-you should be able to understand most of it just by knowing what you do about C++.

WIDGETS (all the "pieces" we are putting together):

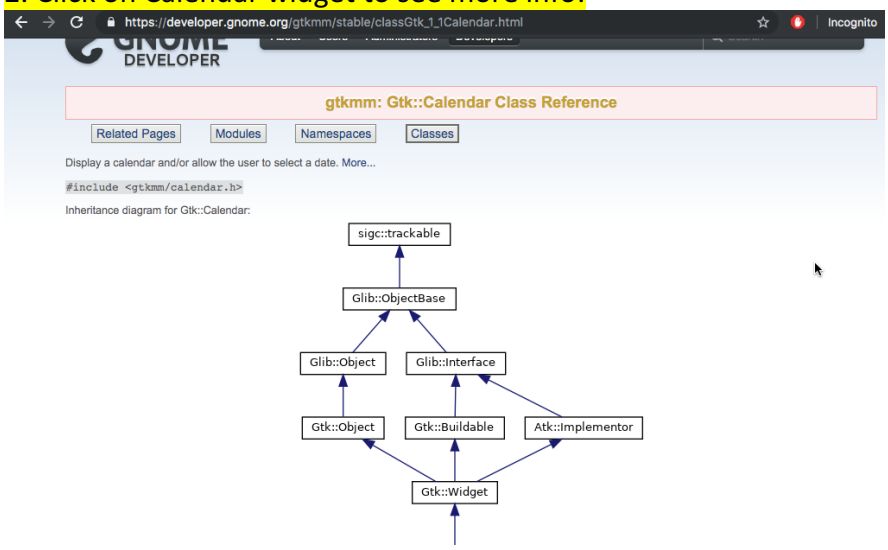
All possible widgets (once again, I won't be doing all of these in class, but with the examples I show you it should be enough to get your started and give you enough knowledge to use any of these):
https://developer.gnome.org/gtkmm/stable/group__Widgets.html
https://developer.gnome.org/gtkmm-tutorial/stable/sec-using-a-gtkmm-widget.html.en
https://developer.gnome.org/gtkmm/stable/classGtk_1_1Widget.html

Example (Calendar):

1. List of all widgets:

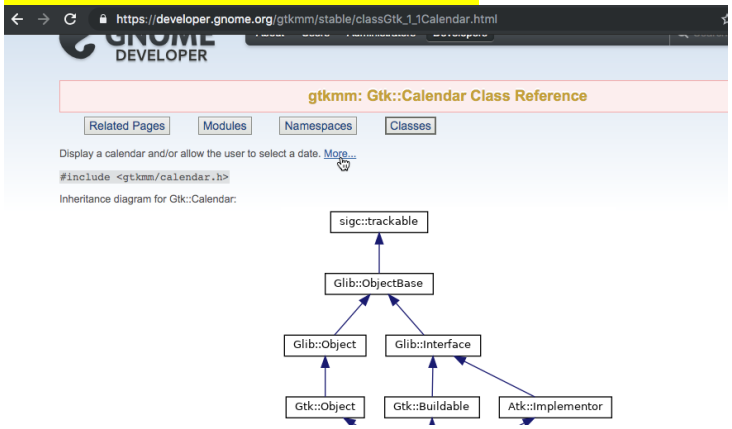


2. Click on Calendar widget to see more info:

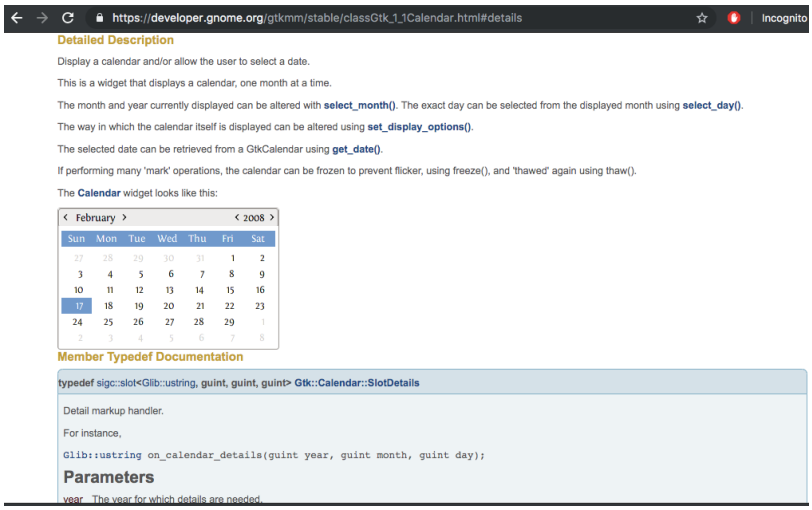


For each of these, you can check the class references to see the available functions etc.

3. Click on more to see more details:



Here you can see more details on putting a calendar in your code:



https://developer.gnome.org/gtkmm/stable/classGtk_1_1Calendar.html#details

Detailed Description

Display a calendar and/or allow the user to select a date.

This is a widget that displays a calendar, one month at a time.

The month and year currently displayed can be altered with **select_month()**. The exact day can be selected from the displayed month using **select_day()**.

The way in which the calendar itself is displayed can be altered using **set_display_options()**.

The selected date can be retrieved from a GtkCalendar using **get_date()**.

If performing many 'mark' operations, the calendar can be frozen to prevent flicker, using freeze(), and 'thawed' again using thaw().

The **Calendar** widget looks like this:

Member Typedef Documentation

typedef sigc::slot<Glib::ustring, guint, guint, guint> Gtk::Calendar::SlotDetails

Detail markup handler.

For instance,

```
Glib::ustring on_calendar_details(guint year, guint month, guint day);
```

Parameters

year The year for which details are needed.

SIGNAL HANDLING:

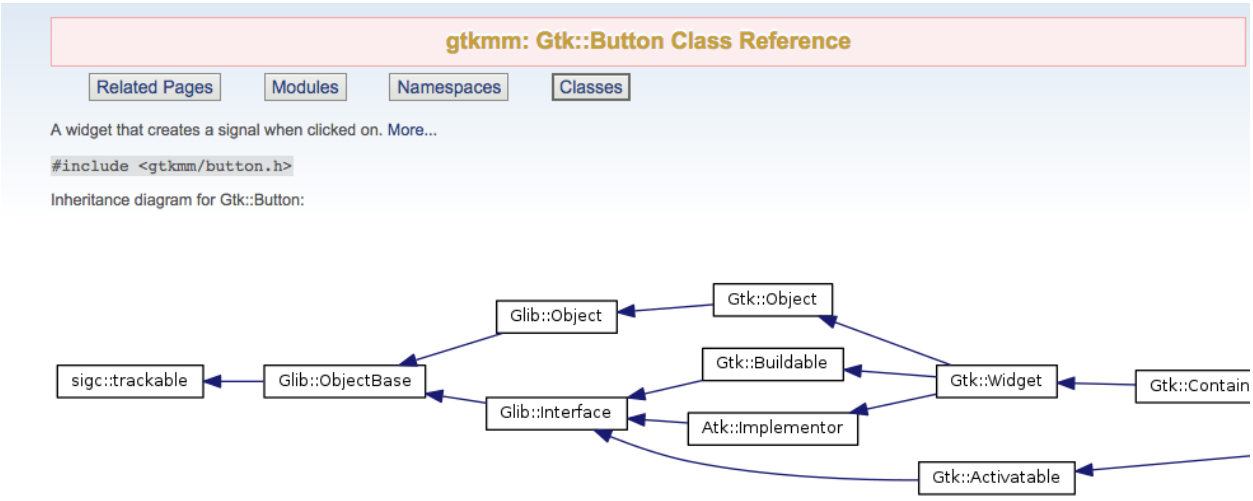
We learned that we can give our buttons some functionality by defining a function to call when it is clicked. But how does it work? (I'm not going to go into the exact implementation details-just the overall flow so you can break down what is going on).

Let's look at something like the following:

button.signal_clicked().connect(sigc::ptr_fun(&on_button_clicked));

1. *button* is our Button Widget. It was created from the Button Class, meaning we can now access functions in the Button class.
2. *signal_clicked()* is a function in the Button class-we know this because *button* is calling it.

At this point, we can tell that whatever the function *signal_clicked()* returns must be something that gives us access to a function called *connect()*. Let's see what *signal_clicked()* actually returns:



gtkmm: Gtk::Button Class Reference

Related Pages | Modules | Namespaces | Classes

A widget that creates a signal when clicked on. More...

```
#include <gtkmm/button.h>
```

Inheritance diagram for Gtk::Button:

bool	**get_always_show_image** () Returns whether the button will ignore the **Gtk::Settings::property_gtk_button_images()** s show the image, if available. More...
Glib::RefPtr< **Gdk::Window** >	**get_event_window** () Returns the button's event window if it is realized, `nullptr` otherwise. More...
Glib::RefPtr< const **Gdk::Window** >	**get_event_window** () const Returns the button's event window if it is realized, `nullptr` otherwise. More...
Glib::SignalProxy< void >	**signal_pressed** ()
Glib::SignalProxy< void >	**signal_released** ()
Glib::SignalProxy< void >	**signal_clicked** ()
Glib::SignalProxy< void >	**signal_enter** ()

This is what is returned by the function

https://developer.gnome.org/gtkmm/stable/classGtk_1_1Button.html#a2fa59b52db13af1de45b2745ea6d70c8

Ok, so we know that something called *Glib::SignalProxy* is calling the function called connect(). Note that *SignalProxy* is in the namespace of something called *Glib*:

Documentation Overview

Getting Started with gtkmm

You might start by reading the **Programming with gtkmm** online book.

Contents of the whole online book.

Selected chapters:

- Installation
- Basics
- Signals
- Container Widgets
- TreeView
- Memory Management
- Glade and Gtk::Builder

API Reference

gtkmm (Gtk::)

- Hierarchy
- Widgets
- Containers
- TreeView
- TextView
- Dialogs
- Menus and Toolbars
- Main loop
- enums and flags
- Stock IDs
- Gtk Namespace

glibmm (Glib::)

- ustring
- Exceptions
- RefPtr<>
- Main Event loop
- Spawning Processes
- Threads
- Miscellaneous Utility Functions
- Character set conversion
- Glib Namespace

giomm (Gio::)

- File
- Streams
- Gio Namespace

https://www.gtkmm.org/en/documentation.html

Glib Namespace Reference

mespaces

namespace	**Ascii**
namespace	**Container_Helpers**
namespace	**Markup**
namespace	**Unicode**

asses

class	**BalancedTree** Balanced Binary Trees — a sorted collection of key/value pairs optimized for searching and traversing in order The **BalancedTree** structure and its associated functions provide a sorted collection of key/value pairs optimized for searching and traversing in order. More...
class	**Checksum** Computes the checksum for data. More...
class	**ConvertError** **Exception** class for charset conversion errors. More...
class	**IConv** Thin iconv() wrapper. More...
class	**Date** Julian calendar date. More...

class	**Cond** An opaque data structure to represent a condition. More...
struct	**StaticPrivate**
class	**Private**
class	**ValueArray** A container structure to maintain an array of generic values. More...
class	**SignalProxyBase**
class	**SignalProxyNormal** The SignalProxy provides an API similar to sigc::signal that can be used to connect sigc::slots to glib signals. More...
class	**SignalProxy0** Proxy for signals with 0 arguments. More...
class	**SignalProxy1** Proxy for signals with 1 arguments. More...
class	**SignalProxy2** Proxy for signals with 2 arguments. More...

http://manual.freeshell.org/glibmm-2.4/reference/html/namespaceGlib.html

You can see *connect()* here:

http://manual.freeshell.org/glibmm-2.4/reference/html/classGlib_1_1SignalProxyNormal.html

https://developer.gnome.org/glibmm/unstable/classGlib_1_1SignalProxy_3_01void_07T_8_8_8_08_4.html#a27b7862ec4b6f24bdb7fc7382bdd89e0

Notice the parameters of *connect()* (above): something called *slot* (usually created by something called *sigc::mem_fun()* or *sigc::ptr_fun()*) and something called *after.*

button.signal_clicked().connect(sigc::ptr_fun(&on_button_clicked));

So now we see that the argument passed for connect is *sigc::ptr_fun(),* so let's see what the argument is and what it returns:

template <class T_return >

pointer_functor0<T_return> sigc::ptr_fun (T_return(*)() _A_func) [inline]

Creates a functor of type **sigc::pointer_functor0** which wraps an existing non-member function.

Parameters:

_A_func Pointer to function that should be wrapped.

Returns:

Functor that executes *_A_func* on invokation.

http://manual.freeshell.org/libsigc++-2.0/reference/html/group__ptr__fun.html#gada8b678665c14dc85eb32d25b7299465

As I always encourage you to do-**LOOK STUFF UP!!!**

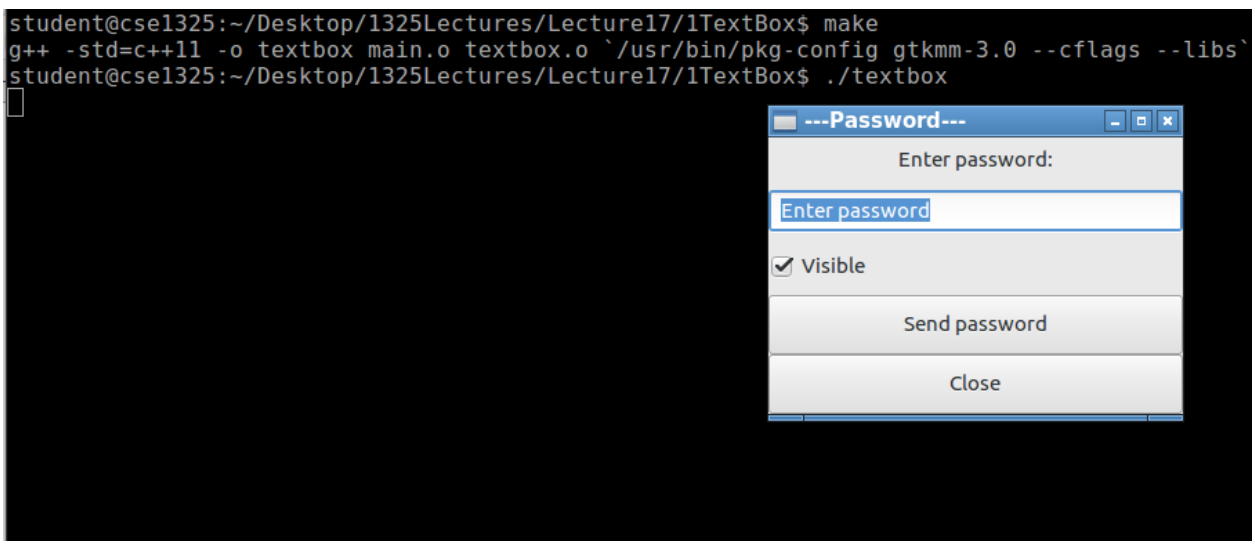
Remember you can't learn everything in a single semester (that's not the purpose of a class)-you are learning the general pieces and tools so when you encounter new situations, you know what something is and what to do with it.

For example:
The *bitoegy* dog *pumpled* over the *crankle* that the *lascarry* rabbit dug.

Even if you don't know what these words mean, you should be able to figure out at least what they are from context!

bitoegy-adjective
pumpled-verb
crankle-noun
lascarry-adjective

Program 1:



textbox.h

```
#ifndef TEXTBOX_H
#define TEXTBOX_H

#include <gtkmm.h>

class Textbox_window : public Gtk::Window
{
public:
  Textbox_window();
  virtual ~Textbox_window();

protected:
  //signal handlers
```

```
  void toggle_checkbox();
  void send_value();
  void close_button();

  //widgets
  Gtk::Box textbox;
  Gtk::Label label;
  Gtk::Box box;
  Gtk::Entry entry;
  Gtk::Button button_close, button_send;
  Gtk::CheckButton checkbutton;
};

#endif
```

textbox.cpp

```
#include "textbox.h"
#include <iostream>

//https://developer.gnome.org/gtk3/stable/gtk3-Standard-Enumerations.html (MORE BELOW)
Textbox_window::Textbox_window() : box(Gtk::ORIENTATION_VERTICAL), button_close("Close"),
  checkbutton("Visible"),
  button_send("Send password")
{
  set_size_request(300, 200);
  set_title("---Password---");

  add(box);

  label.set_text("Enter password:");
  box.pack_start(label);

  entry.set_max_length(50);
  entry.set_text("Enter password");
  entry.select_region(0, entry.get_text_length());
  box.pack_start(entry);

  box.pack_start(textbox);

  textbox.pack_start(checkbutton);

  checkbutton.signal_toggled().connect(sigc::mem_fun(*this,
          &Textbox_window::toggle_checkbox) );
  checkbutton.set_active(true);


  button_send.signal_clicked().connect(sigc::mem_fun(*this,
          &Textbox_window::send_value));
  box.pack_start(button_send);
```

```
  button_close.signal_clicked().connect( sigc::mem_fun(*this,
          &Textbox_window::close_button));
  box.pack_start(button_close);

  button_close.set_can_default();
  button_close.grab_default();

  show_all_children();
}

Textbox_window::~Textbox_window()
{
}

void Textbox_window::send_value()
{
  std::string input=entry.get_text();


  //enum (MESSAGE_INFO) http://transit.iut2.upmf-grenoble.fr/doc/gtkmm-
                                                    3.0/reference/html/namespaceGtk.html#enum-members

  Gtk::MessageDialog dialog(*this, "Password sent!",false,Gtk::MESSAGE_INFO);
  dialog.set_secondary_text(input);
  dialog.run();
}

void Textbox_window::toggle_checkbox()
{
  entry.set_visibility(checkbutton.get_active());
}

void Textbox_window::close_button()
{
  hide();
}
```

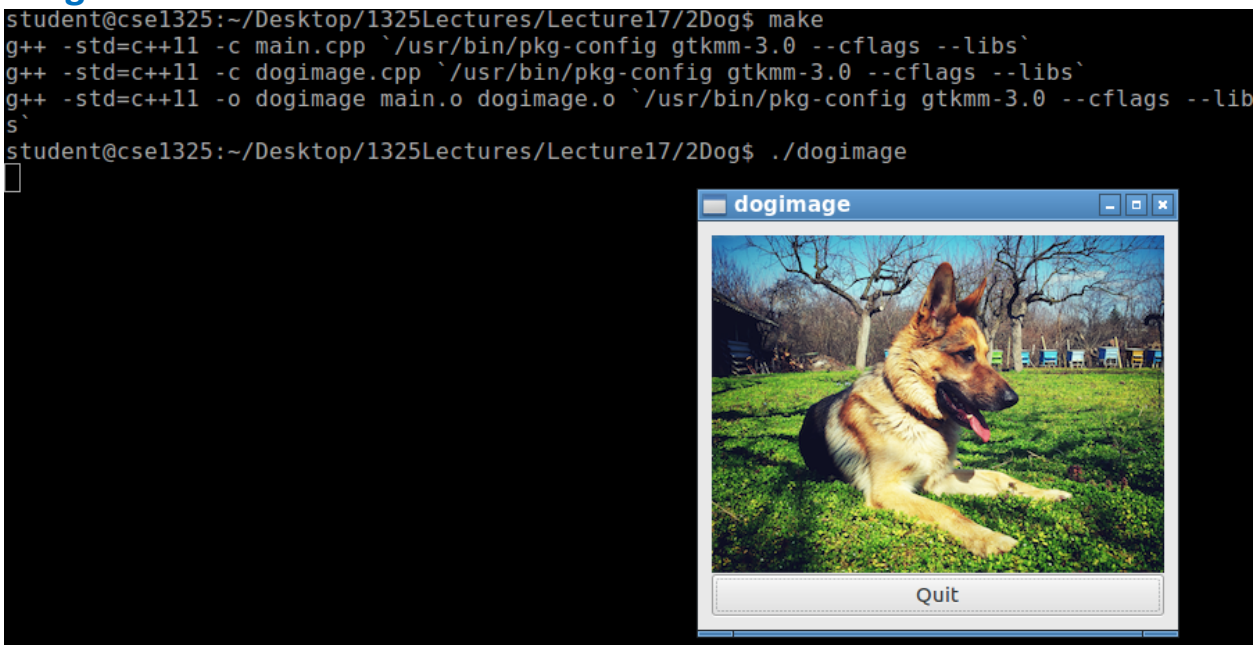
main.cpp

```
#include "textbox.h"
#include <gtkmm/application.h>

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

  Textbox_window window;
```

```
  Gtk::Main::run(window);
  return 0;
}
```

Note: I kept pics in the same folder

Program 2:



dogimage.h

```
#ifndef DOG_H
#define DOG_H

#include <gtkmm.h>

//the class Dog_window inherits from Gtk::Window
class Dog_window : public Gtk::Window
{
  //constructor and destructor
public:
  Dog_window();
  virtual          ~Dog_window();

protected:

  //widgets
  Gtk::Image          image;
  Gtk::Button         quit;
  Gtk::Grid           grid;
};


#endif
```

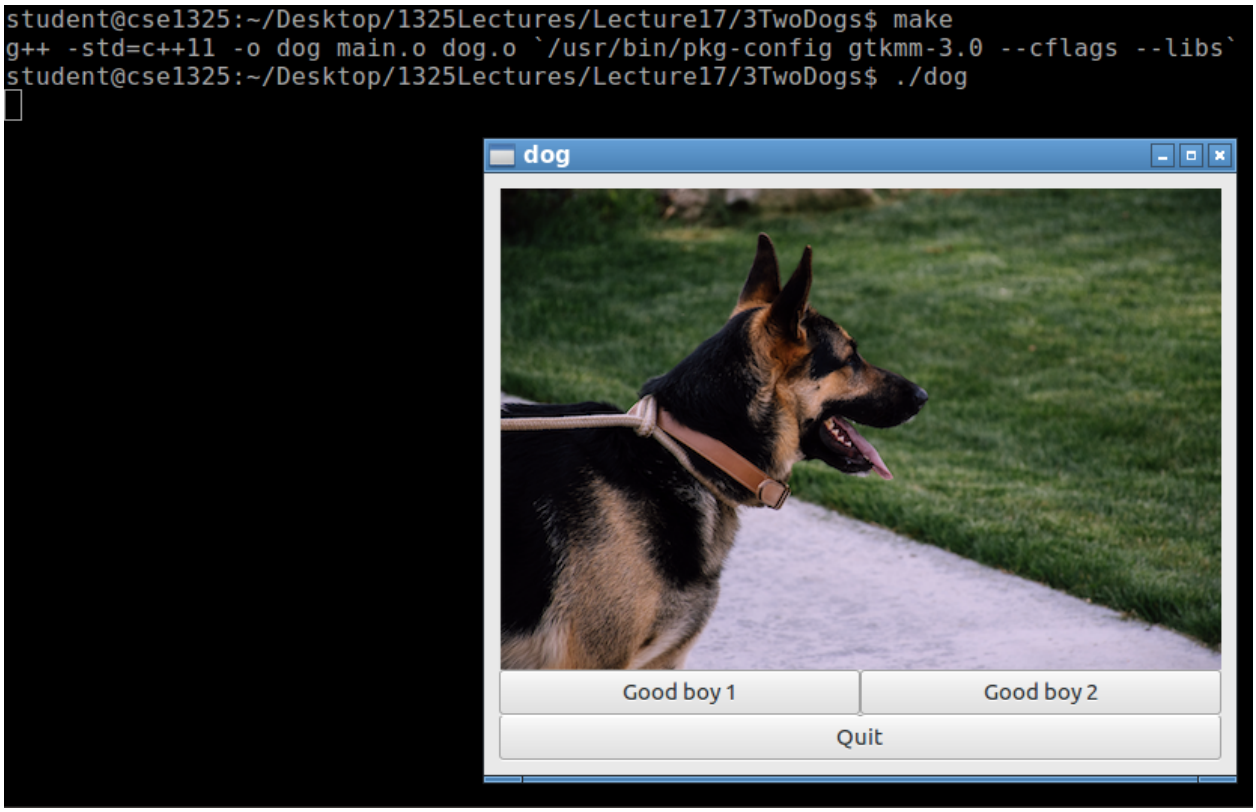
dogimage.cpp

```
#include "dogimage.h"

Dog_window::Dog_window()
{
 //set window border
 this->set_border_width(10);

 //set image (make sure to include picture in the folder)
 image.set("bestbreedever.png");
 grid.attach(image,0,0,1,1);

 quit.add_label("Quit");
 quit.signal_pressed().connect(sigc::mem_fun(*this,&Dog_window::close));
 grid.attach(quit,0,2,1,1);

 grid.show_all();

 //add the main grid (where we put all the stuff above) to the window
 add(grid);
}

//destructor
Dog_window::~Dog_window()
{}
```

main.cpp

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


int main(int argc, char* argv[])
{
 // this line initializes gtkmm (starts it in the program)
 Gtk::Main app(argc, argv);

 Dog_window w;

 Gtk::Main::run(w);
 return 0;
}
```

Program 3:



dog.h

```
#ifndef TWO_IMAGE_H
#define TWO_IMAGE_H

#include <gtkmm.h>


// inherits from Gtk::Window
class Two_image : public Gtk::Window
{
   // Constructor and destructor
public:
   Two_image();
   virtual          ~Two_image();

protected:
   //signal handlers
   void              display_image1();
   void              display_image2();

   //widgets:
   Gtk::Image          image;
   Gtk::Button       button_image1, button_image2, quit;
   Gtk::Grid         grid;
};
```

```
#endif
```

dog.cpp

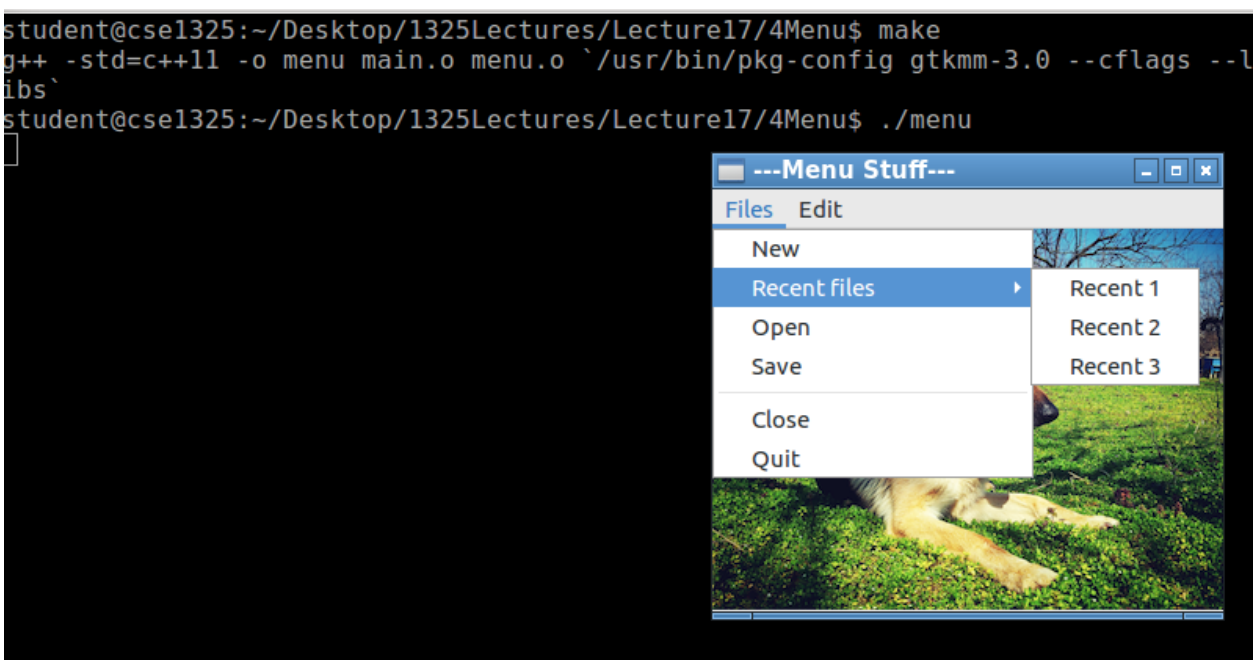
```
#include "dog.h"
#include <iostream>


// constructor
Two_image::Two_image()
{
    set_border_width(10);

   //show image-make sure to put the image in the folder
   image.set("bestbreedever.png");
   grid.attach(image,0,0,2,1);

   //handle first button
   button_image1.add_label("Good boy 1");
   button_image1.signal_pressed().connect(sigc::mem_fun(*this,&Two_image::display_image1));
   grid.attach(button_image1,0,1,1,1);

   //handle second button
   button_image2.add_label("Good boy 2");
   button_image2.signal_pressed().connect(sigc::mem_fun(*this,&Two_image::display_image2));
   grid.attach(button_image2,1,1,1,1);

   //handle quit button
   quit.add_label("Quit");
   quit.signal_pressed().connect(sigc::mem_fun(*this,&Two_image::close));
   grid.attach(quit,0,2,2,1);

   grid.show_all();

   //add grid to window
   add(grid);
}


//destructor
Two_image::~Two_image()
{}

//change image to first image-make sure to put the image in the folder
void Two_image::display_image1()
{
   image.set("bestbreedever.png");
}
```

```
//change image to second image-make sure to put the image in the folder
void Two_image::display_image2()
{
   image.set("bestdogbreed2.png");
}
```

main.cpp

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


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

        Two_image dogwindow;

        Gtk::Main::run(dogwindow);

        return 0;
}
```

Program 4:



menu.h

```
#ifndef MENU_H
#define MENU_H

#include <gtkmm.h>

class Menu_window : public Gtk::Window
{
public:
  Menu_window();
  virtual ~Menu_window();


  //widgets
  Gtk::Image image;
  Gtk::Menu submenu1, menu_recent;
  Gtk::MenuBar menubar;
   Gtk::MenuItem menufiles, menufiles1,new_file,recent_file, open, save, close, quit, recent1, recent2,
recent3;
    Gtk::SeparatorMenuItem line;
  Gtk::Box box;

};

#endif
```

menu.cpp

```
#include "menu.h"
#include <iostream>

Menu_window::Menu_window()
: box(Gtk::ORIENTATION_VERTICAL)
{
  set_size_request(300, 200);
  set_title("---Menu Stuff---");

  add(box);

  box.pack_start(menubar);
  menufiles.set_label("Files");
  menubar.append(menufiles);

  menufiles1.set_label("Edit");
  menubar.append(menufiles1);

  menufiles.set_submenu(submenu1); //submenu for files on menubar

  new_file.set_label("New"); //add this to submenu
  submenu1.append(new_file);
```

```
  recent_file.set_label("Recent files"); //add this to submenu
  submenu1.append(recent_file);

  recent1.set_label("Recent 1");
  recent2.set_label("Recent 2");
  recent3.set_label("Recent 3");

  recent_file.set_submenu(menu_recent); //create submenu for Recent files

  menu_recent.append(recent1); //add to submenu for Recent files
  menu_recent.append(recent2); //add to submenu for Recent files
  menu_recent.append(recent3); //add to submenu for Recent files

  open.set_label("Open"); //add this to submenu
  submenu1.append(open);

  save.set_label("Save"); //add this to submenu
  submenu1.append(save);

  submenu1.append(line); //add line before close and quit menu items

  close.set_label("Close"); //add this to submenu
  submenu1.append(close);

  quit.set_label("Quit"); //add this to submenu
  submenu1.append(quit);

  image.set("bestbreedever.png");
  box.pack_start(image);


  show_all_children();
}

Menu_window::~Menu_window()
{
}
```

main.cpp

```
#include "menu.h"
#include <gtkmm/application.h>

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

  Menu_window window;

  Gtk::Main::run(window);
```

```
  return 0;
}
```