CSE 1325: Object-Oriented Programming

Lecture 13 – Chapters 12 and 16
(Using gtkmm)

Basic GUIs and Dialogs

Mr. George F. Rice
george.rice@uta.edu

Based on material by Bjarne Stroustrup
www.stroustrup.com/Programming

ERB 402
Office Hours:
Tuesday Thursday 11 - 12
Or by appointment

Lecture 12

Quick Review

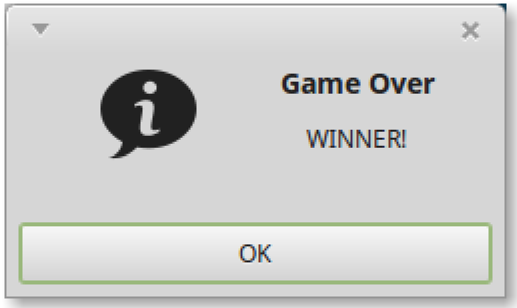
- Put the following user interface technologies in chronological order of introduction: Voice, GUI, Touch / Gesture, CLI, Punch Card, Paper Tape
  Paper Tape, Punch Card, CLI, GUI, Voice, Touch / Gesture
  - How do web apps fit in? Similar to voice time frame, but maturing slowly
- What is the Principle of Least Astonishment? "A user interface component should behave as the users expect it to behave."
- A pointer variable contains the **memory address** of the value of interest. Accessing the value of interest via the pointer value is called **dereferencing**.
- Memory for the value of interest for pointer variables is usually allocated from the **heap** using the **new** keyword, and freed using the **delete** keyword.
- Access a member of an object via a pointer uses the **->** operator.
- The **Façade** pattern implements a simplified interface to a complex class or package.

Lecture 12
Quick Review

- What is the primary philosophical difference between CLI and GUI applications?With CLI, the program controls the sequence of events. With GUI, the user controls the sequence of events.
- Why is the main program loop part of gtkmm rather than written by you? It is difficult to write correctly, and rarely varies
- To ensure your gtkmm program is compiled and linked using the right libraries, use the **make** tool with a **Makefile**.
- How is memory allocated from the stack? With a “normal” declaration How (and when) is it subsequently deallocated? Automatically, when the variable goes out of scope
- How is memory allocated from the heap? Using the “new” keyword How (and when) is it subsequently deallocated?
  Only when explicitly deallocated using the delete keyword
- When are the two variants of “delete” used? Use “delete” to deallocate a simple variable on the heap, and “delete[ ]” to deallocated a vector or array

Overview: GUIs and Dialogs

- Inheritance – Living la Vida OO
- “Hello, World” in GUI Land
- Pango, or Pseudo-HTML
- Writing a Dialogs Class
  - Message
  - Input
  - Question
  - Image
- Converting a CLI to Dialogs
  - Guessing Game Example


Game Over

WINNER!

https://developer.gnome.org/gtkmm/stable/

Concise “PIE” Definition

of Object-Oriented Programming

Polymorphism	We'll cover this later!
+ Inheritance	We'll cover this now!
<u>+ Encapsulation</u>	We covered this!
Object-Oriented Programming	

Review

Inheritance

- **Inheritance** – Reuse and extension of fields and method implementations from another class

Expertise
/eks-per-tyz/
def: Special skill or knowledge in a particular subject, eg. He has expertise in his field of molecular science

- The original class is called the **base class** (e.g., exception)
- The extended class is called the **derived class** (e.g., Bad_area)

"Assets"

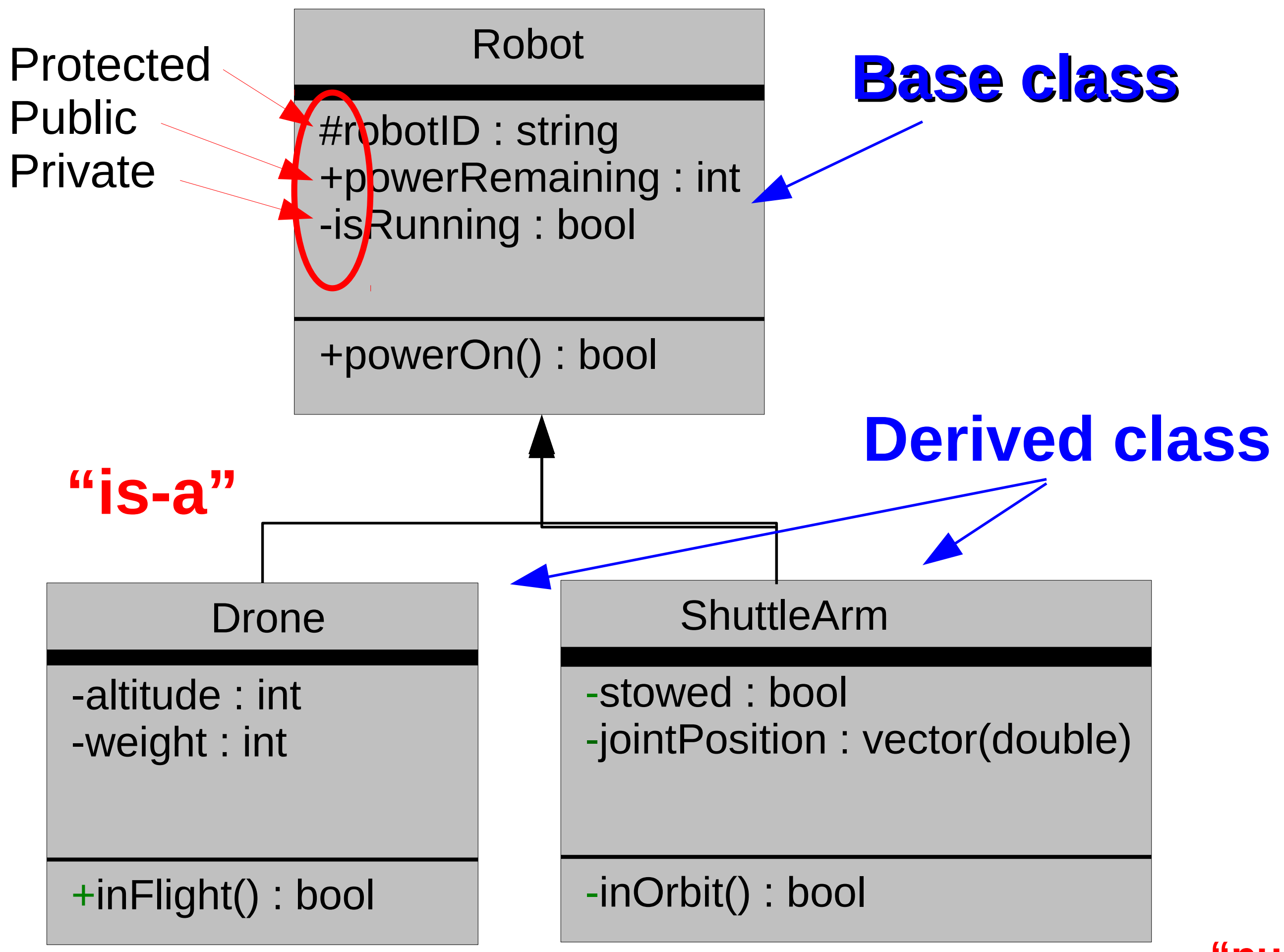
The Heir → The Ancestor

```
class Bad_area : public exception {
class View     : public DrawingArea {
```

Derived Class **Base Class**

Review

Class Hierarchy



```
class Robot {
```

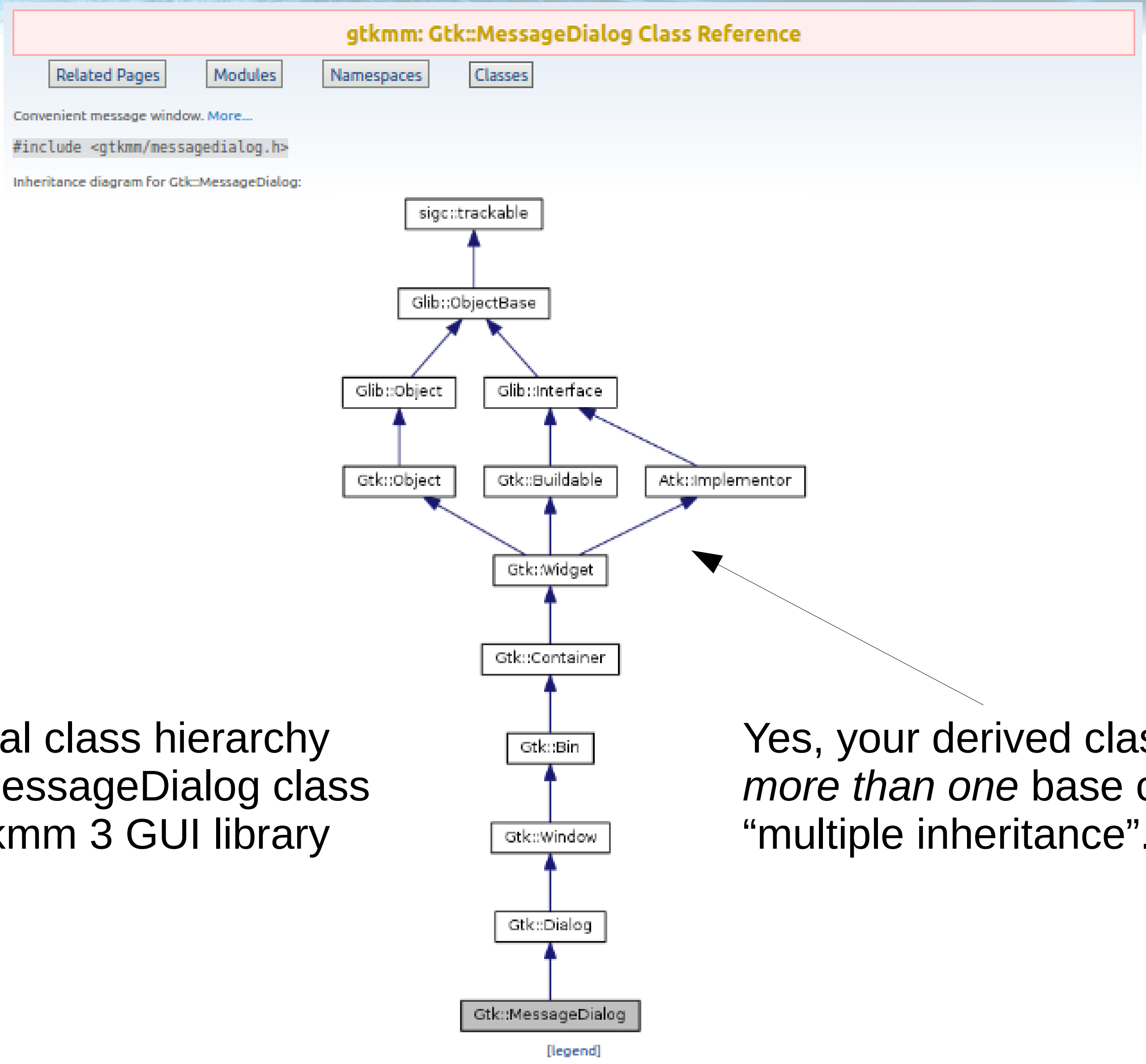
All 3 have robotID, isRunning, powerRemaining, and powerOn().

ONLY Robot has isRunning (because it's private)

```
class Drone :
        public Robot {

class ShuttleArm :
        public Robot {
```

"public" means that all public fields in Robot will be public in ShuttleArm. "private" would make public fields in Robot private in ShuttleArm.

Class Hierarchies are Key to GUI Libraries


gtkmm: Gtk::MessageDialog Class Reference

Convenient message window. More...

`#include <gtkmm/messagedialog.h>`

Inheritance diagram for Gtk::MessageDialog:

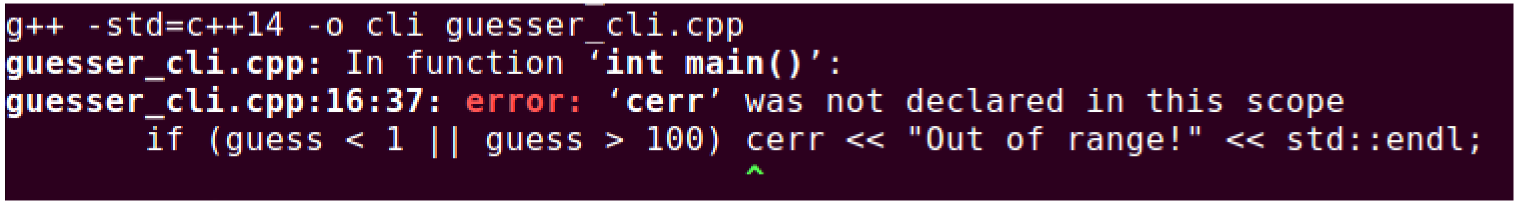
The actual class hierarchy for the MessageDialog class in the gtkmm 3 GUI library

Yes, your derived class can have *more than one* base class, called “multiple inheritance”.

https://developer.gnome.org/gtkmm/stable/classGtk_1_1MessageDialog.html

Programming like Programmers

- `using namespace std;` is rather bad
  - It imports a LOT of names into our default namespace
  - It causes collisions and ambiguity when we inadvertently reuse a name, e.g., std::to_string vs our own to_string
- Professionals generally don’t do this
- We’ll stop now (mostly)
  - Instead, we’ll prefix <u>all</u> members of std with std::
  - Common culprits include std::string, std::vector, std::cout, std::cin, std::cerr, std::exception, and std::runtime_error
  - The compiler will notify you thus if you forget:



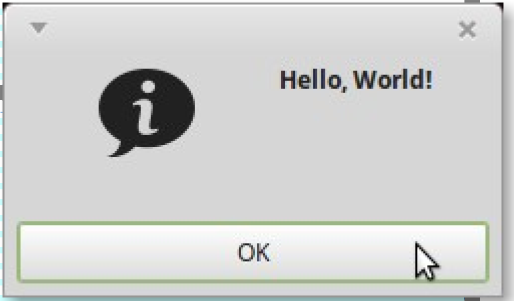
```
g++ -std=c++14 -o cli guesser_cli.cpp
guesser_cli.cpp: In function ‘int main()’:
guesser_cli.cpp:16:37: error: ‘cerr’ was not declared in this scope
       if (guess < 1 || guess > 100) cerr << "Out of range!" << std::endl;
                                     ^
```

Review

Writing “Hello, World” in gtkmm

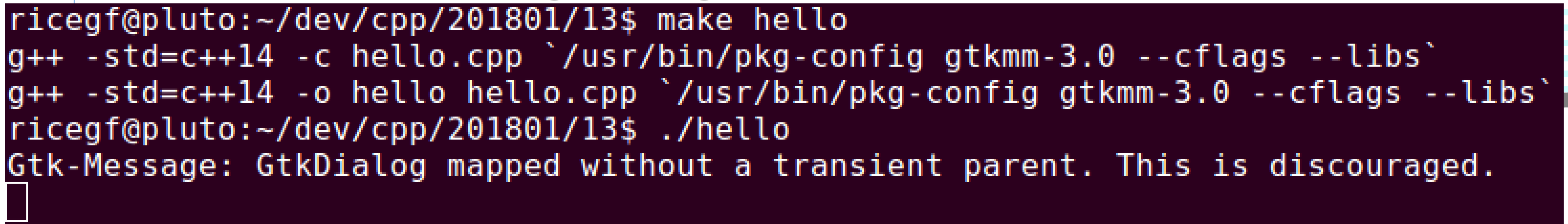
main.cpp

```
#include <gtkmm.h>

int main(int argc, char *argv[])
{
    // Initialize GTK for dialogs – we’ll use a factory for apps
    Gtk::Main kit(argc, argv);

    // Create a simple dialog containing "Hello, World!"
    Gtk::MessageDialog *dialog = new Gtk::MessageDialog{"Hello, World!"};

    // Turn control over to gtkmm until the user clicks OK
    dialog->run();
}
```



Makefile

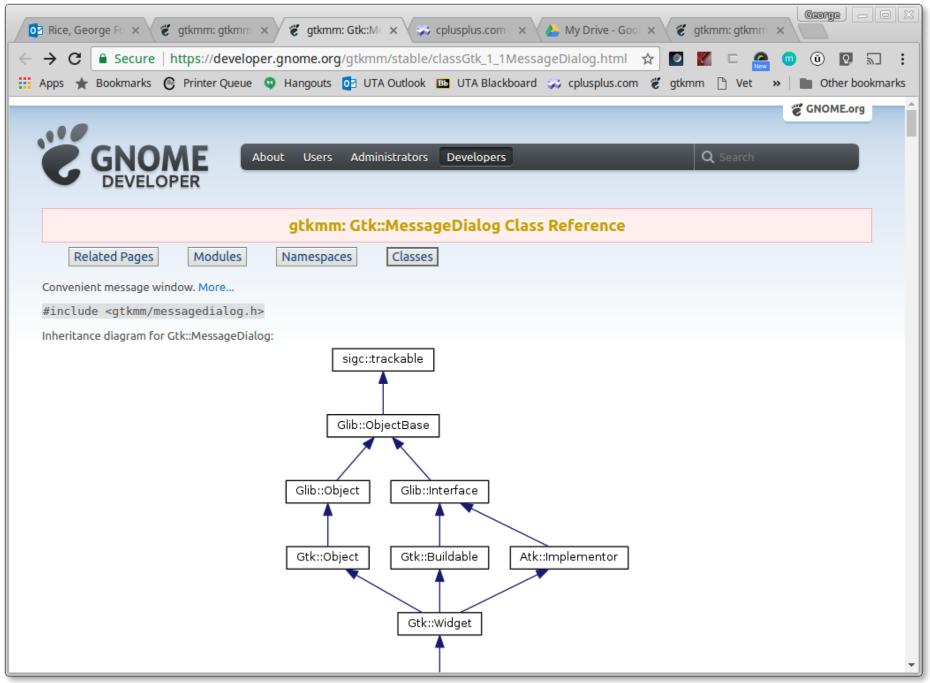
```
CXXFLAGS += -std=c++14
GTKFLAGS = `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`

hello: hello.o
        $(CXX) $(CXXFLAGS) -o hello hello.cpp $(GTKFLAGS)
hello.o: hello.cpp
        $(CXX) $(CXXFLAGS) -c hello.cpp $(GTKFLAGS)

clean:
        -rm -f *.o *.gch *~ gui cli test
```



```
ricegf@pluto:~/dev/cpp/201801/13$ make hello
g++ -std=c++14 -c hello.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
g++ -std=c++14 -o hello hello.cpp `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
ricegf@pluto:~/dev/cpp/201801/13$ ./hello
Gtk-Message: GtkDialog mapped without a transient parent. This is discouraged.
```

How to Learn More About Gtk::MessageDialog

- Look it up on the Gnome website
  - https://developer.gnome.org/gtkmm/stable/

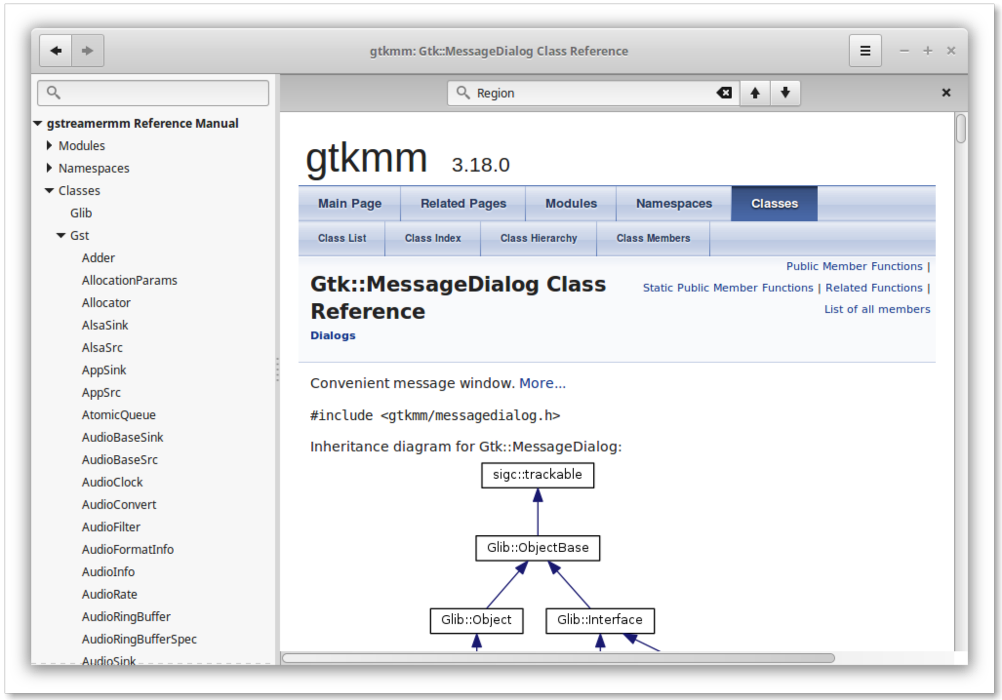


Convenient message window. More...

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

Inheritance diagram for Gtk::MessageDialog:

- Use "devhelp" locally (if installed)



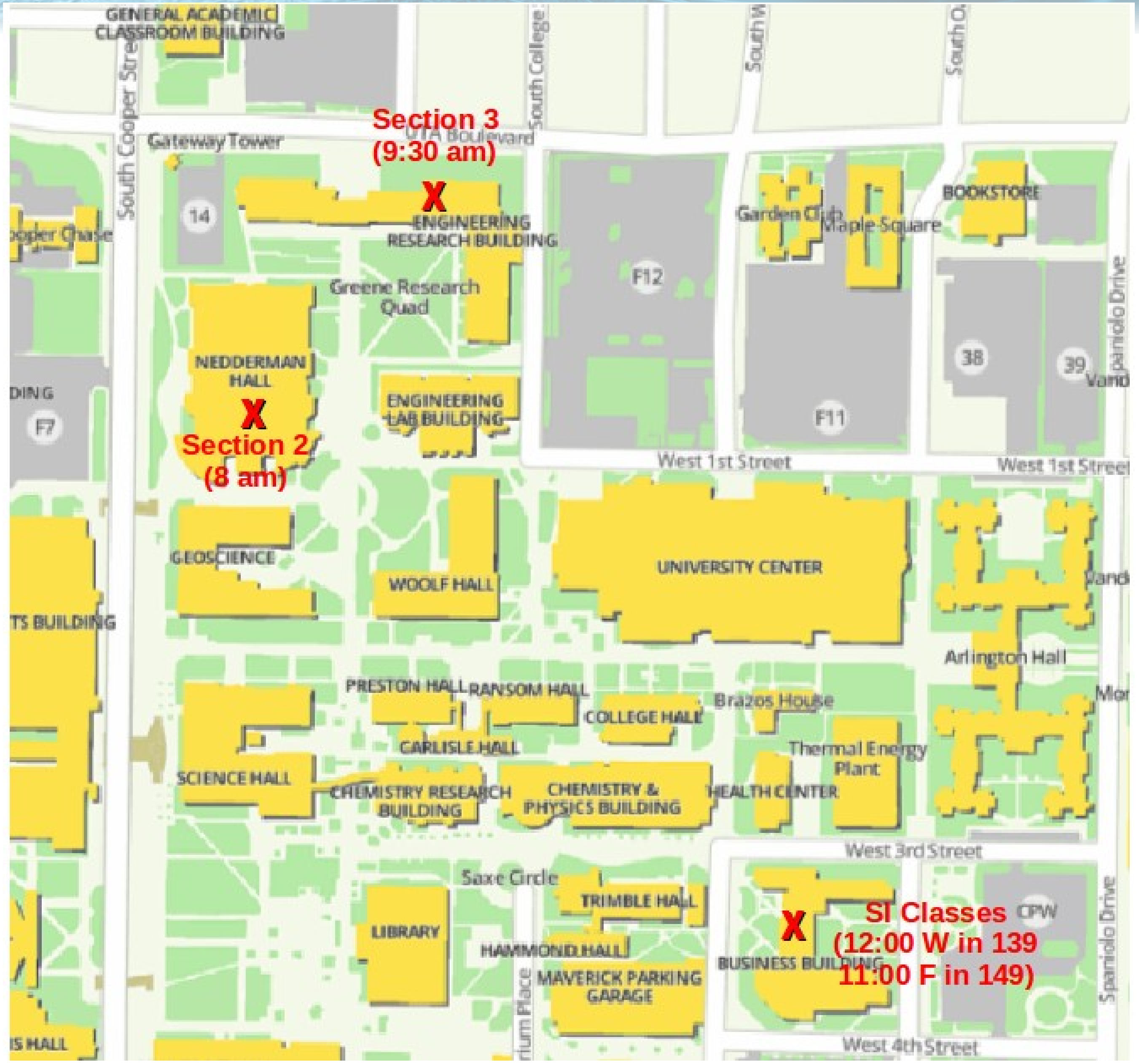
Convenient message window. More...

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

Inheritance diagram for Gtk::MessageDialog:

```
sudo apt-get install libgtkmm-3.0-doc
sudo apt-get install libgstreamermm-1.0-doc
sudo apt-get install devhelp
```

Learn about gtkmm Docs *tomorrow* at the SI Session!



Interesting MessageDialog Members

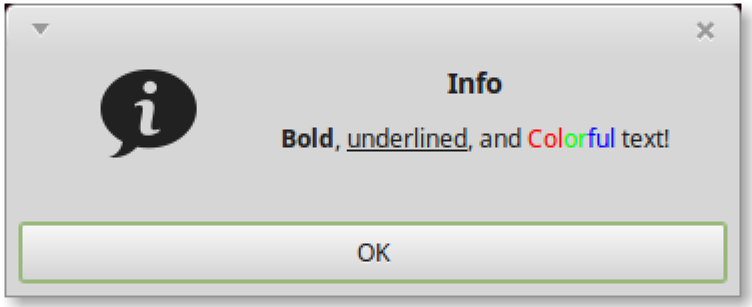
- `MessageDialog (`
  ```
  const Glib::ustring& message,
  bool use_markup=false,
  MessageType type=MESSAGE_INFO,
  ButtonsType buttons=BUTTONS_OK,
  bool modal=false)
  ```

  Note the default values enable us to construct a MessageDialog instance with as few as one parameter!

  – **ustring** is gtkmm’s Unicode version of std::string, with full conversions to and from std::string (hint: just use “string” and all will be well)
  – **use_markup** determines whether **Pango** markup is interpreted from the message (more on Pango shortly)
  – **type** determines the **icon** to be presented: MESSAGE_INFO, _WARNING, _QUESTION, _ERROR, _OTHER
  – **buttons** determines which buttons to display: BUTTONS_NONE, _OK, _CLOSE, _CANCEL, _YES_NO, or _OK_CANCEL
  – **modal** is true if no other dialogs may take focus while this dialog is open, false otherwise (hint: This should almost ALWAYS be false!)

Pango Markup

- Pango works similar to HTML
  - The root tag is effectively <span>, which requires a </span> close and accepts attributes such as:
    - font, font_size, font_style, font_weight, etc.
    - fgcolor and alpha, bgcolor and bgalpha
    - underline, strikethrough, and their _color variants
  - Convenience tags: <b>, <big>, <i>, <s>, <sub>, <sup>, <small>, <tt> (monospace), and <u>
  - "<b>Bold</b>, <u>underlined</u>,
    and <span fgcolor='#ff0000'>Col</span>
    <span fgcolor='#00ff00'>or</span>
    <span fgcolor='#0000ff'>ful</span> text!"
- The Pango reference manual is at https://developer.gnome.org/pango/stable/

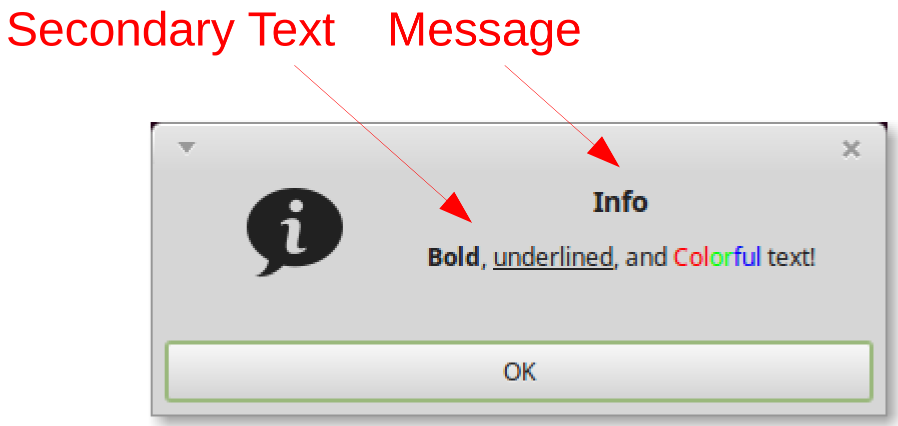


Bold, underlined, and Colorful text!

See test_dialogs.cpp

Interesting MessageDialog Members

- `set_secondary_text ( const Glib::ustring& text, bool use_markup=false )`
  - Secondary text is positioned under the message
    - In effect, the message becomes the title and the secondary text the message
  - use_markup is true if the text should be processed for Pango tags



Info

Bold, <u>underlined</u>, and Colorful text!

Writing a Dialogs Class

- We can define a Dialogs class with a convenience method, “message”
  - This enables single-line message dialogs

```
#ifndef _DIALOGS_H
#define _DIALOGS_H
#include <iostream>
#include <gtkmm.h>

class Dialogs {
  public:
    static void message(
        std::string msg,
        std::string title = "Info");

#endif
```

```
void Dialogs::message(std::string msg, std::string title) {
    Gtk::MessageDialog *dialog = new Gtk::MessageDialog(title);
    dialog->set_secondary_text(msg, true);
    dialog->run();

    dialog->close();
    while (Gtk::Main::events_pending())  Gtk::Main::iteration();

    delete dialog;
}
```

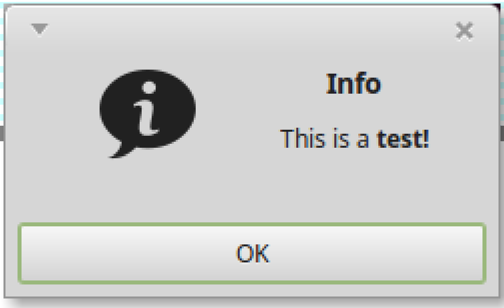
The while loop forces the dialog to process all pending events.
Without it, the dialog hangs around like a bad penny.
This isn’t a problem in real apps, so don’t sweat it (NOT on the exam!)

Writing a Dialogs Class

- The makefile is quite simple, as is its use

```
CXXFLAGS += -std=c++14
GTKFLAGS = `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`

test: test_dialogs.o dialogs.o *.h
        $(CXX) $(CXXFLAGS) -o test test_dialogs.cpp dialogs.o $(GTKFLAGS)
        ./test 2> /dev/null

test_dialogs.o: test_dialogs.cpp *.h
        $(CXX) $(CXXFLAGS) -c test_dialogs.cpp $(GTKFLAGS)

dialogs.o: dialogs.cpp *.h
        $(CXX) $(CXXFLAGS) -c dialogs.cpp $(GTKFLAGS)

clean:
        -rm -f *.o *.gch *~ test
```

Makefile


Info

This is a **test!**

```
#include “dialogs.h”                // Include the class
int main {
    Gtk::Main kit(argc, argv);   // Initialize gtkmm
    Dialogs::message("This is a test!");
}
```

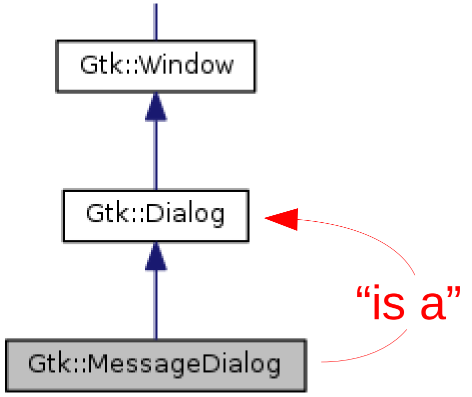
main.cpp

Expanding the Dialogs Class

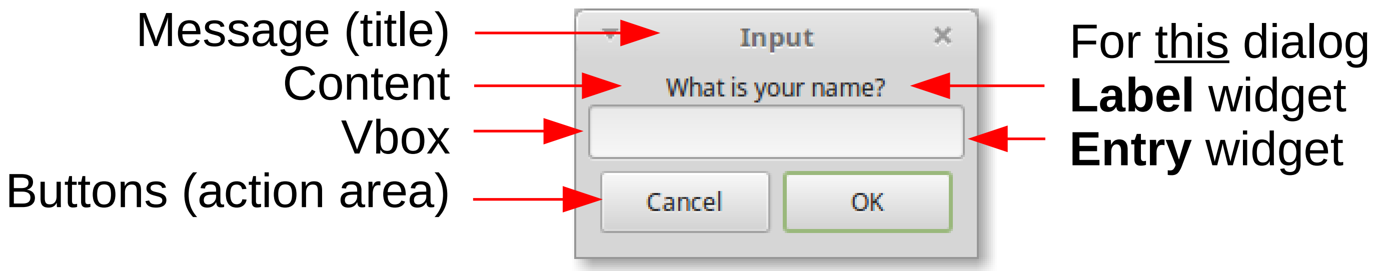
- We’ll need more than a cout equivalent – we also need to cin!
  - We’ll call this method “input”
- To take advantage of our GUI environment
  - We’ll add a “question” method that presents a message, and offers one or more buttons
  - Since we’re doing graphics, we’ll also add an “image” dialog that displays an image from disk

Dialog

- MessageDialog is a derived case
  - The base class is Dialog
  - MessageDialog inherits from Dialog



- Dialog is a pre-built Gtk::Window
  - It provides a message (title), a Content area, a Vbox area, and a Button (action) area
  - You can put any number of widgets into Content and Vbox that you like – they are "containers"



Adding Buttons to a Dialog

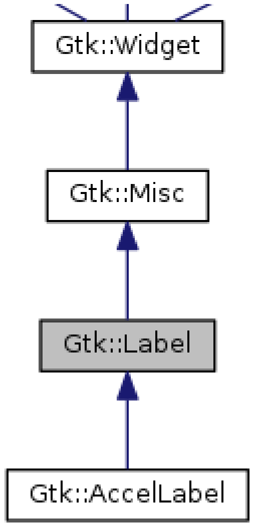
- Dialog makes special provision for this
- dialog.add_button(const Glib::ustring& button_text, int response_id)
  - button_text is the text on the button, of course
  - response_id is a unique integer that will be returned when a button is clicked
- Dialog.set_default_response(int response_id)
  - Sets the default button, which is usually activated when the user just presses Enter

Adding Widgets to a Dialog

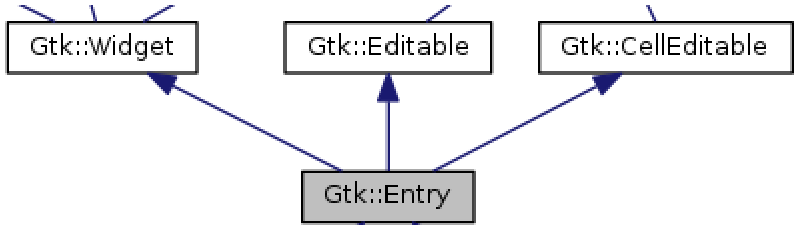
- `Gtk::`*`Widget`*` *widget = new Gtk::`*`Widget`*`{ }`
  - This instances a new widget on the heap
  - Replace *Widget* with a subclass, e.g., Entry
- Configure the widget as needed
- *`widget`*`->show()`
  - This makes the widget visible on-screen
- Finally, “pack” the widget into one of the containers
  - `dialog->get_vbox()->pack_start(*widget)`
  - `dialog->get_content_area()->pack_start(*widget)`

The Label Widget

- Gtk::Label displays read-only text
- `Label (const Glib::ustring& label, bool mnemonic=false)`
  - Label is the text to display
  - Mnemonic can enable “keyboard shortcuts”, e.g., Alt-C (copy)
- `Label->set_use_markup(bool setting=true)`
  - Enables Pango (similar to HTML) in the label



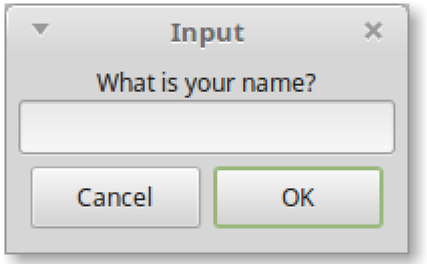
The Entry Widget



- Gtk::Entry accepts a single line of text from the user
- Entry { }
  - `entry->set_max_length(int)` sets width in chars
  - `entry->set_text()` sets the default text for user to edit
  - `entry->get_text()` reads the text the user entered
    - The dialog returns the Response_ID of the button pushed
    - Then use `entry->get_text()` to find out what the user typed
- Pango is <u>not</u> supported

Creating a Text Input Dialog

- We will use all 4 areas
  - Set the text (title)
  - Create a Label for the message
    - e.g., “What is your name?”
  - Create an Entry to accept the text and provide it to our method
  - Add 2 buttons, “Cancel” and “OK”



- Return the text from the Entry instance if OK is pressed, or a special “cancel text” otherwise
  - An exception could also be thrown here

Writing the Input Dialog

dialogs.h

```
    // A request for a line of text input
    static string input(string msg, string title = "Input", string default_text = "",
                 string cancel_text = "CANCEL");
```

dialogs.cpp

```
std::string Dialogs::input(std::string msg, std::string title,
                           std::string default_text, std::string cancel_text) {
    Gtk::Dialog *dialog = new Gtk::Dialog();
    dialog->set_title(title);

    Gtk::Label *label = new Gtk::Label(msg);
    dialog->get_content_area()->pack_start(*label);
    label->show();

    dialog->add_button("Cancel", 0);
    dialog->add_button("OK", 1);
    dialog->set_default_response(1);

    Gtk::Entry *entry = new Gtk::Entry{};
    entry->set_text(default_text);
    entry->set_max_length(50);
    entry->show();
    dialog->get_vbox()->pack_start(*entry);

    int result = dialog->run();
    std::string text = entry->get_text();

    dialog->close();
    while (Gtk::Main::events_pending())  Gtk::Main::iteration();
```

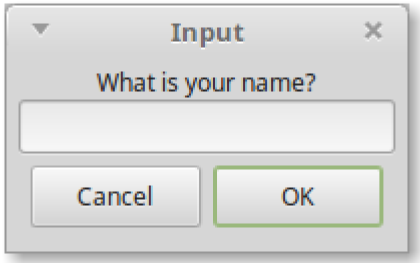
```
    delete entry;
    delete label;
    delete dialog;

    if (result == 1)
        return text;
    else
        return cancel_text;
}
```

Testing Text Input

- The Makefile is unchanged

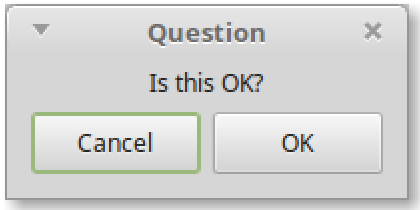
```
#include “dialogs.h”                 // Include the class
#include <iostream>                  // For cout
int main {
    Gtk::Main kit(argc, argv);  // Initialize gtkmm
    std::cout << Dialogs::input("What is your name?") << std::endl;
}
```

main.cpp



Creating a Question Dialog

- We will use only 3 areas
  - Set the text (title)
  - Create a Label for the message
    - e.g., “Is this OK?”
  - Add any number of buttons using text supplied by a vector of string
    - By default display “Cancel” and “OK”
- Return the index of the button actually clicked

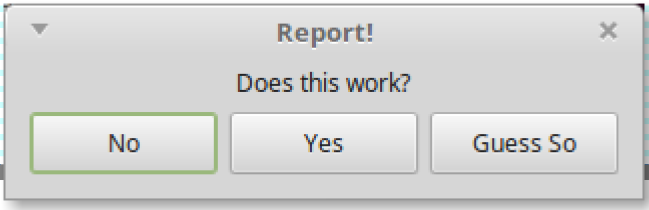


Writing the Question Dialog

```
    // A question is a message that allows the user to respond with a button
    static int question(std::string msg, std::string title = "Question",
           std::vector<std::string> buttons = {"Cancel", "OK"});
```

dialogs.h

```
int Dialogs::question(std::string msg, std::string title,
         std::vector<std::string> buttons) {
    Gtk::Dialog *dialog = new Gtk::Dialog();
    dialog->set_title(title);

    Gtk::Label *label = new Gtk::Label(msg);
    dialog->get_content_area()->pack_start(*label);
    label->show();

    for(int i=0; i<buttons.size(); ++i) dialog->add_button(buttons[i], i);

    int result = dialog->run();

    dialog->close();
    while (Gtk::Main::events_pending())  Gtk::Main::iteration();

    delete label;
    delete dialog;

    return result;
}
```



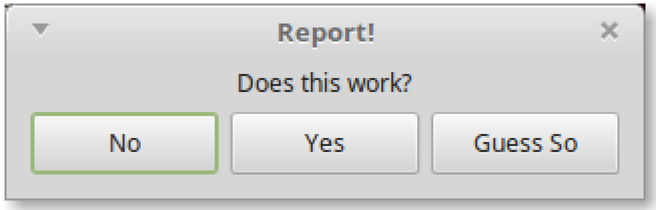
dialogs.cpp

Testing the Question Dialog

- The makefile is unchanged

```
#include “dialogs.h”                 // Include the class
#include <iostream>                  // For cout
int main {
    Gtk::Main kit(argc, argv);  // Initialize gtkmm

    // Define text for each button
    std::vector<std::string> buttons = {"No", "Yes", "Guess So"};
    std::cout << Dialogs::question(
        "Does this work?",
        "Report!",
        buttons) << std::endl;
}
```

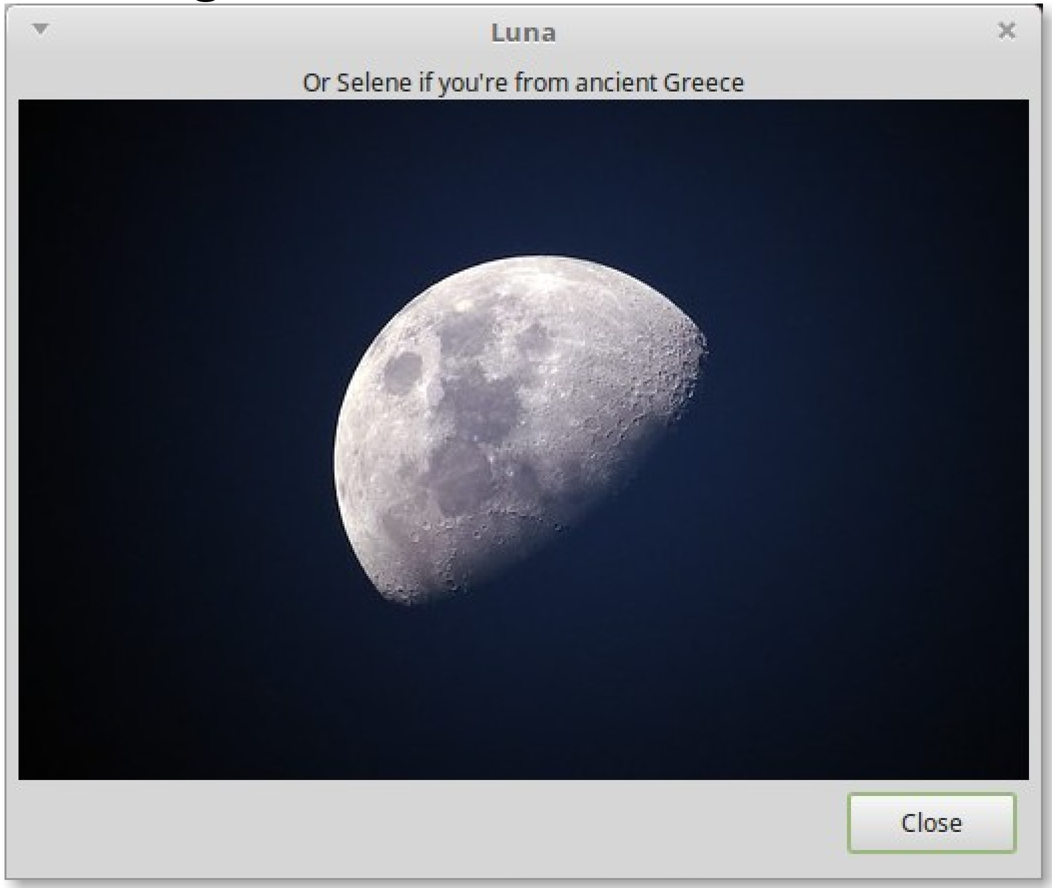
main.cpp



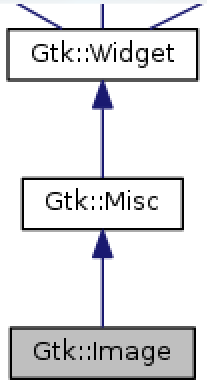
Does this work?

Creating an Image Dialog

- We will use all 4 areas
  - Set the text (title), e.g., “Luna”
  - Create a Label for the message
    - e.g., “Or Selene if you’re from ancient Greece”
  - Add an Image widget to load and display an image file
  - Add a “Close” button



The Image Widget



- Image simply displays an image
  - `image {const std::string& file}` loads the image from the specified file
  - Static and animated images can also be displayed from memory using other constructors

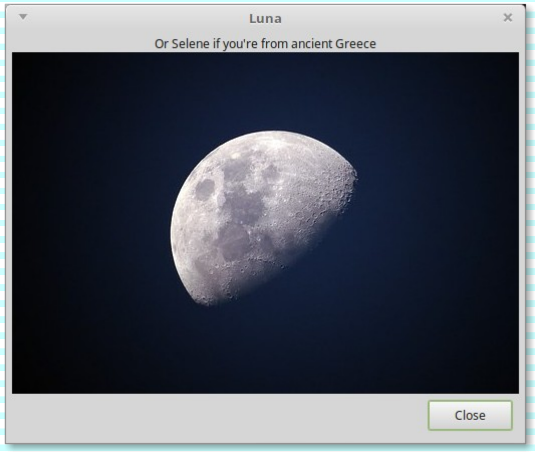
https://developer.gnome.org/gtkmm/stable/classGtk_1_1Image.html

Creating an Image Dialog

dialogs.h

```
// Display an image from a disk file
static void image(std::string filename, std::string title = "Image", std::string msg= "");
```

```
void Dialogs::image(string filename, string title, string msg) {
    Gtk::Dialog *dialog = new Gtk::Dialog();
    dialog->set_title(title);

    Gtk::Label *label = new Gtk::Label(msg);
    dialog->get_content_area()->pack_start(*label);
    label->show();

    dialog->add_button("Close", 0);
    dialog->set_default_response(0);

    Gtk::Image *image = new Gtk::Image{filename};
    image->show();
    dialog->get_vbox()->pack_start(*image);

    int result = dialog->run();

    dialog->close();
    while (Gtk::Main::events_pending())  Gtk::Main::iteration();

    delete image;
    delete label;
    delete dialog;

    return;
}
```

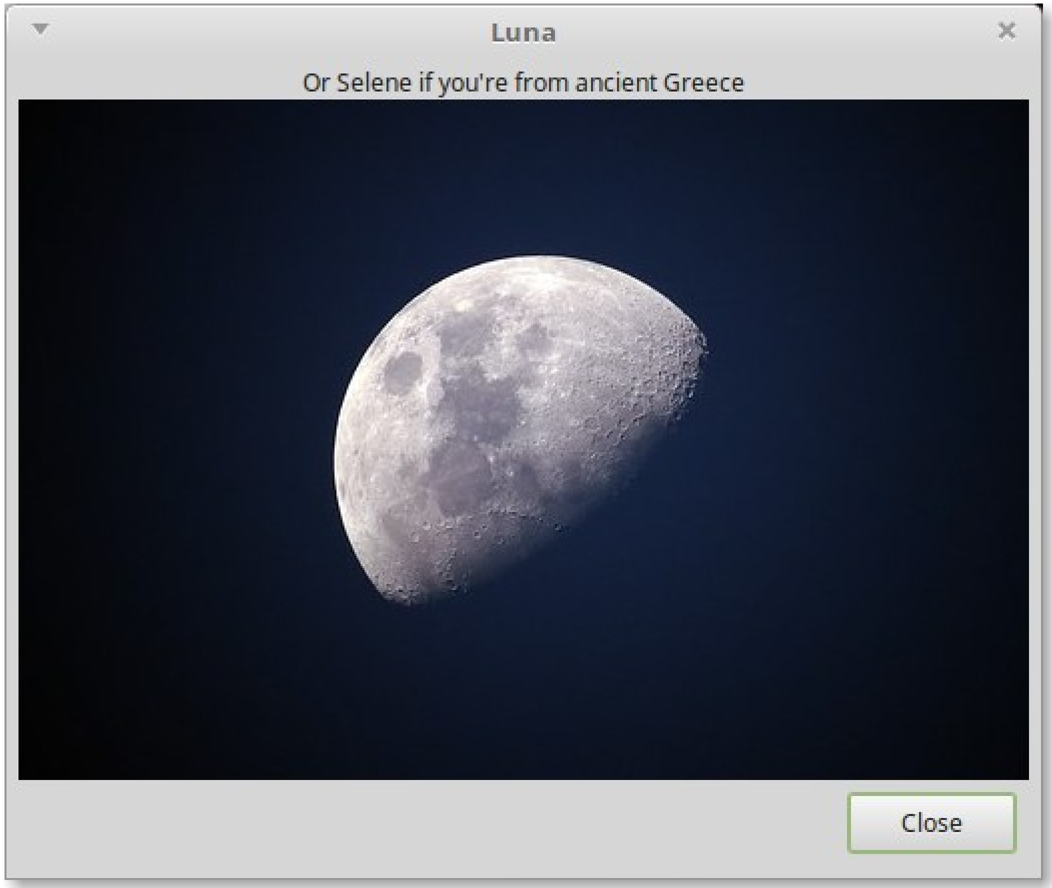


dialogs.cpp

Testing the Image Dialog

main.cpp

```
#include “dialogs.h”                 // Include the class
#include <iostream>                  // For cout
int main {
    Gtk::Main kit(argc, argv);    // Initialize gtkmm
    Dialogs::image("moon.jpg", "Luna", "Or Selene if you're from ancient Greece");
}
```


Luna

Or Selene if you're from ancient Greece

Obtaining Dialogs

- The Dialogs class is available on Blackboard attached to this lecture
- Dialogs will help you *a lot* with Homework #6 / Sprint #2 of the Library Management System
  - Recommended (but not strictly required)

Example Migrating CLI to Dialogs

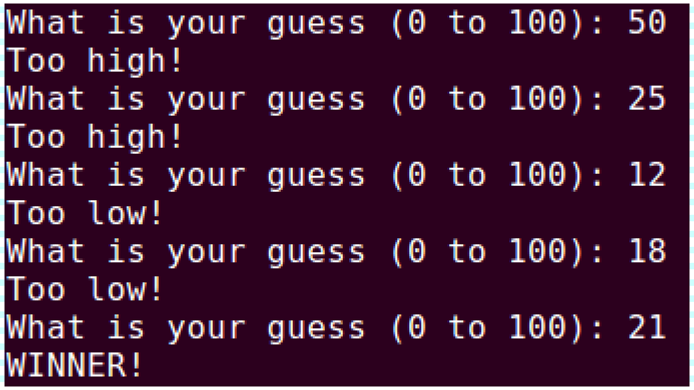
- Given Dialogs, it’s fairly straightforward to convert a CLI program to a “GUI”
  - This isn’t a “real” GUI, though, because it doesn’t have a main window
  - We’ll use a trivial guessing game as an example

CLI Guessing Game

to be converted to GUI dialogs

guesser_cli.cpp

```cpp
#include <iostream>
#include <random>

int main() {
    int num;             // The number to be guessed
    int guess = 0;       // The user's guess
    std::string text; // Temp for holding user's input

    srand ( time(NULL) );
    num = rand() % 100 + 1;

    while (num != guess) {
      cout << "What is your guess (0 to 100): ";
      getline(cin, text);
      guess = stoi(text);
      if (guess < 1 || guess > 100) cerr << "Out of range!" << endl;
      else if (guess > num) cout << "Too high!" << endl;
      else if (guess < num) cout << "Too low!" << endl;
      else cout << "WINNER!" << endl;
    }
}
```



```
What is your guess (0 to 100): 50
Too high!
What is your guess (0 to 100): 25
Too high!
What is your guess (0 to 100): 12
Too low!
What is your guess (0 to 100): 18
Too low!
What is your guess (0 to 100): 21
WINNER!
```

GUI Dialogs Guessing Game

Page 1 of 2

guesser_gui.cpp

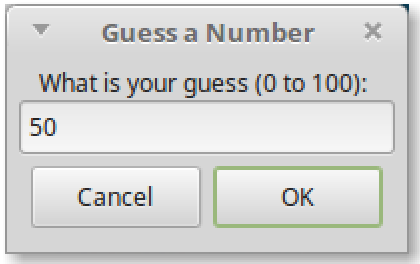
```
// No longer required
// #include <iostream>
#include <random>

// Add dialogs (and implicitly gtkmm)
#include "dialogs.h"

using namespace std;

int main(int argc, char *argv[]) {
    int num;            // The number to be guessed
    int guess = 0;      // The user's guess
    std::string text; // Temp for holding user's input

    srand ( time(NULL) );
    num = rand() % 100 + 1;

    // Initialize gtkmm and add a title
    Gtk::Main kit(argc, argv);
    std::string title = "Guess a Number";
```

GUI Dialogs Guessing Game

Page 2 of 2

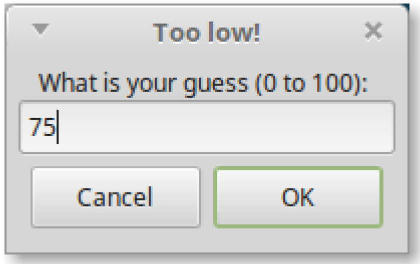
guesser_gui.cpp

```
    while (num != guess) {

      // cout << "What is your guess (0 to 100): ";
      // getline(cin, text);
      text = Dialogs::input("What is your guess (0 to 100): ", title);
      if (text == "CANCEL") break;

      guess = stoi(text);

      // if (guess < 1 || guess > 100) cerr << "Out of range!" << endl;
      // else if (guess > num) cout << "Too high!" << endl;
      // else if (guess < num) cout << "Too low!" << endl;
      // else cout << "WINNER!" << endl;
      if (guess < 1 || guess > 100) title = "Out of range!";
      else if (guess > num) title = "Too high!";
      else if (guess < num) title = "Too low!";
      else Dialogs::message("WINNER!", "Game Over");
    }
}
```


Guess a Number

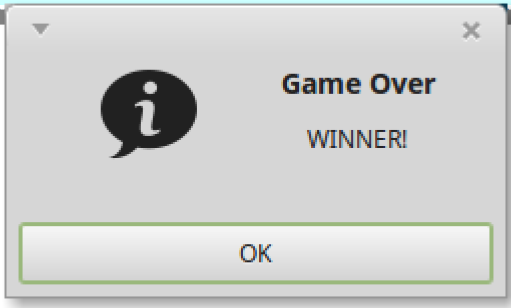
What is your guess (0 to 100):

50


Too low!

What is your guess (0 to 100):

75


Game Over

WINNER!

Dialog and MessageDialog (and Dialogs) are Façades

- Real GUI programming is significantly more complex than this
  - That’s why we start with these façades :-)
- We’ll cover how to implement a full GUI program next class
  - Main window, with menus, tool bars, etc.
  - Secondary windows and custom dialogs

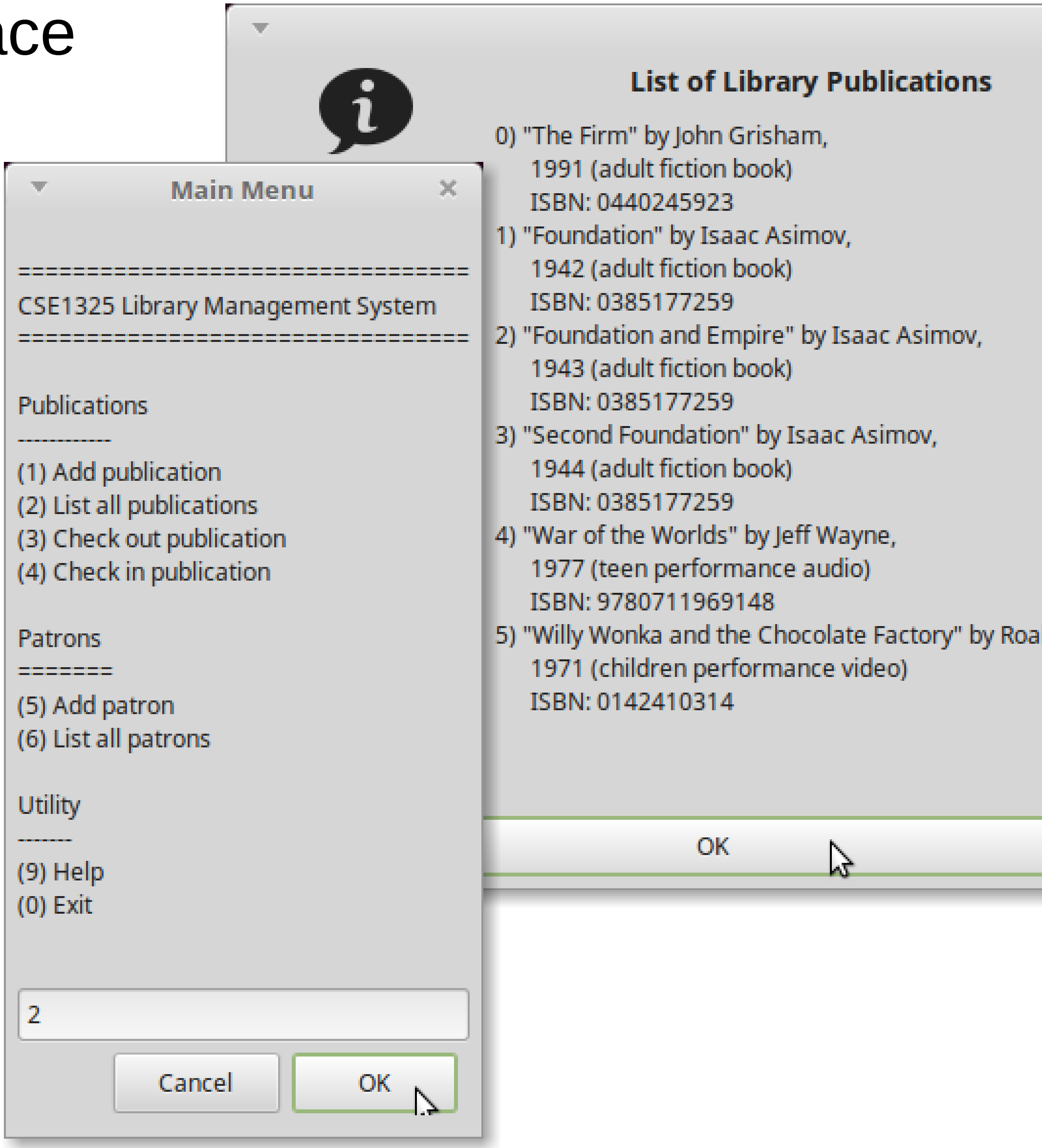
Quick Review

- Each program using gtkmm must include the header file ________.
- True or False: \`/usr/bin/pkg-config gtkmm-3.0 --cflags –libs\` need only be added to the linker line, not the compiler lines, in the Makefile.
- True or False: The Dialogs class must be instanced before the methods can be called.

Homework #6

Sprint 2 of 3

- Rework your LMS user interface with **dialogs**
  - NO console I/O at all – dialogs ONLY!
  - Rich text at the bonus level
  - A true custom dialog at the extreme bonus level
- As always, use git for version management
- Manage *all 3 sprints* via the Scrum spreadsheet
- Details are on Blackboard
- **Due Thurs, March 8 at 8 am**



```
================================
CSE1325 Library Management System
================================

Publications
------------
(1) Add publication
(2) List all publications
(3) Check out publication
(4) Check in publication

Patrons
=======
(5) Add patron
(6) List all patrons

Utility
-------
(9) Help
(0) Exit

2
```

```
List of Library Publications
0) "The Firm" by John Grisham,
   1991 (adult fiction book)
   ISBN: 0440245923
1) "Foundation" by Isaac Asimov,
   1942 (adult fiction book)
   ISBN: 0385177259
2) "Foundation and Empire" by Isaac Asimov,
   1943 (adult fiction book)
   ISBN: 0385177259
3) "Second Foundation" by Isaac Asimov,
   1944 (adult fiction book)
   ISBN: 0385177259
4) "War of the Worlds" by Jeff Wayne,
   1977 (teen performance audio)
   ISBN: 9780711969148
5) "Willy Wonka and the Chocolate Factory" by Roa
   1971 (children performance video)
   ISBN: 0142410314
```

Next Class

- Review chapter 13 and 14 in Stroustrop
  - Remember he is using FLTK and a façade!
  - Do the drills
- Attend the SI session (BOFA 149 at 11 am Friday) to learn how to cruise the gtkmm documentation!
- We continue Graphical User Interfaces
  - Custom dialogs
  - Main window
  - Observer and Factory Patterns