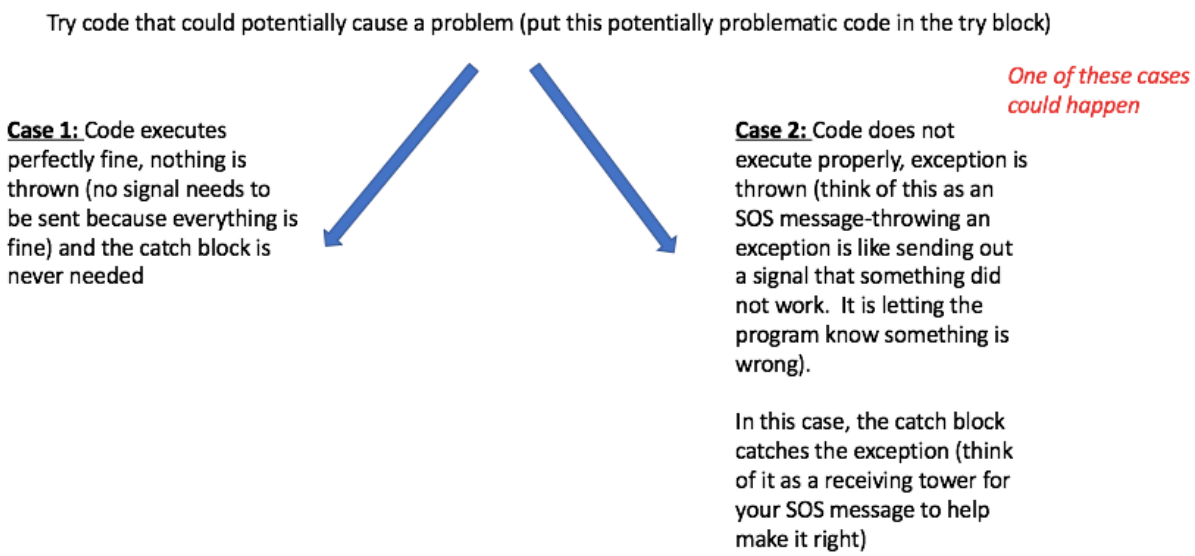
Exception Handling

We use exceptions to handle any potential problem code we may encounter. This can be something that will crash our program, give an undesired outcome that does not make sense in our program or cause some other unintended outcome in our program. In C++, we can make our own exception class (example below).

Note 1: I won't discuss this in class but if you are curious you can look up a programming technique called RAII (Resource Acquisition Is Initialization)

Note 2: A commonly practiced rule is to "throw by value, catch by reference"-We will see more about this in a future lecture



Example 1:

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Floating point exception: 8
```

```
#include <iostream>

int main(int argc, char **argv) {

   int a=10, b=0;
   int c;

   c=a/b; //error from dividing by 0 (see above for error)
}
```

Using exception handling:

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Can't divide by 0
```

```
#include <iostream>

int main(int argc, char **argv) {

   int a=25, b=0;
   int c;

   //put the potential trouble code inside the try
   try{
     if(b==0) //can't do this
     {
       throw "Can't divide by 0..."; //go to catch block-don't go to next line c=a/b;
     }
     c=a/b;

   }

   //catch exception thrown above (“Cant divide by 0”) is caught and then printed to screen
   catch(string e){
     std::cout<<e<<std::endl;
   }
}
```

Step 1:

```
#include <iostream>

int main(int argc, char **argv) {

   int a=25, b=0;
   int c;

try{
    if(b==0)
    {
      throw "Can't divide by 0...";
    }
    c=a/b;

  }

    catch(string e)
  {
    std::cout<<e<<std::endl;
  }
}
```

Try code that could potentially cause a problem (put this potentially problematic code in the try block)

Step 2:

```
#include <iostream>

int main(int argc, char **argv) {

   int a=25, b=0;
   int c;

try{
    if(b==0)
    {
      throw "Can't divide by 0...";
    }
    c=a/b;

  }

    catch(string e)
  {
    std::cout<<e<<std::endl;
  }
}
```

Since b==0, we will throw an exception (SOS message that something went wrong)-Case 2

Step 3:

```
#include <iostream>

int main(int argc, char **argv) {

   int a=25, b=0;
   int c;

try{
     if(b==0)
     {
       throw "Can't divide by 0...";
      }
     c=a/b;

   }

    catch(string e)
   {
    std::cout<<e<<std::endl;
   }
}
```

We now catch that exception (receiving that SOS message so we can do something about it-in this case, let the user know the problem with the code)

Example 2:

```
computer$ ./a.out
How much money to give your friend?
-50
Can't give your friend negative amount!!!
```

```
#include <iostream>

using namespace std;

int main(int argc, char **argv) {

   int cash;

   cout<<"How much money to give your friend?"<<endl;
   cin>>cash;

   //make sure they dont try to give negative $$$
   try{
     if(cash<0) //can't give negative $$$
     {
       throw (cash);  //this negative amount will be thrown to the catch (like hot potato)
     }

     else //gave a good amount! program is fine
     {
       cout<<"Your friend appreciates the money!"<<endl;
     }
```

```
  }

  catch(int& n)
  {
    cout<<n<<" is negative! Can't give this."<<endl;
  }
}
```

Note: whatever you throw, your catch needs to have the same type. In the first example, I threw a string so I caught a string. In this example, notice I threw an integer, so I'm catching an integer.

Example (*multiple catch blocks*):

One exception (index):

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Enter letter to put in array:
i
Enter letter to put in array:
o
Enter letter to put in array:
j
Exception: index 3 is out of range
```

Another exception (letter not allowed):

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Enter letter to put in array:
e
Enter letter to put in array:
a
a is not allowed.
```

```
#include <iostream>

using namespace std;

int main(int argc, char **argv) {

 char letter;
 try
 {
  char * mystring;
  mystring = new char [3];
  if (mystring == NULL)
  {
    throw "Allocation failure";  //second catch block
  }

  for (int i=0; i<=50; i++)
  {
    if (i>2) //this is the highest index
    {
      throw i; //first catch block-throw int that causes exception
```

```
    }
    cout<<"Enter letter to put in array:"<<endl;
    cin>>letter;

      if(letter=='a') //don't allow a! third catch block
      {
        throw letter;
      }
      mystring[i]=letter;
    }

  }

 catch (int i)
 {
  cout << "Exception: ";
  cout << "index " << i << " is out of range" << endl;
 }
 catch (char * str)
 {
  cout << "Exception: " << str << endl;
 }

 catch(char c)
 {
  cout << c<< " is not allowed." << endl;
 }

 return 0;
}
```

Notes:

1. If you throw an exception but don't have a catch, you will end up with an error:

```
computer$ ./a.out
libc++abi.dylib: terminating with uncaught exception of type char
Abort trap: 6
```

```
#include <iostream>

int main()
{
   try  {
     throw 'f';
   }
   catch (int x)
   {
      std::cout << "Caught an int."<<std::endl;
   }
```

```
    //we don't have a catch for a char
}
```

2. … means you can catch all types of exceptions (in the previous program, it will skip over the *int* catch block and go to the *catch all* block:

```
#include <iostream>

int main()
{
   try  {
     throw 'f';
   }
   catch (int x)
   {
      std::cout << "Caught an int."<<std::endl;
   }

   catch (...) //this should come at the end (think of it like a final case scenario)
   {
      std::cout << "Catch all..."<<std::endl;
   }

}
```

Exception class example (Making a class):

We have a class called exception and derived classes for specific exceptions (we will see in a future lectures examples using these pre-made derived classes-see below for some of the derived classes). We can also inherit from this class and make our own exception class (see below).

absolute beginners

The C Standard Library

- The C Standard Library

The C++ Standard Library

- C++ Library - Home
- C++ Library - <fstream>
- C++ Library - <iomanip>
- C++ Library - <ios>
- C++ Library - <iosfwd>
- C++ Library - <iostream>
- C++ Library - <istream>
- C++ Library - <ostream>
- C++ Library - <sstream>
- C++ Library - <streambuf>
- C++ Library - <atomic>
- C++ Library - <complex>
- C++ Library - <exception>

It is a standard exception class. All objects thrown by components of the standard library are derived from this class. Therefore, all standard exceptions can be caught by catching this type by reference.

Declaration

Following is the declaration for std::exception.

```
class exception;
```

Example

In below example for std::exception.

```
#include <thread>
#include <vector>
#include <iostream>
#include <atomic>

std::atomic_flag lock = ATOMIC_FLAG_INIT;

void f(int n) {
   for (int cnt = 0; cnt < 100; ++cnt) {
      while (lock.test_and_set(std::memory_order_acquire))
         ;
      std::cout << "Output from thread " << n << '\n';
      lock.clear(std::memory_order_release);
   }
}
```

Exception class is kept in the standard library

https://www.tutorialspoint.com/cpp_standard_library/exception.htm

Derived types (scattered throughout different library headers)

bad_alloc	Exception thrown on failure allocating memory (class)
bad_cast	Exception thrown on failure to dynamic cast (class)
bad_exception	Exception thrown by unexpected handler (class)
bad_function_call C++11	Exception thrown on bad call (class)
bad_typeid	Exception thrown on typeid of null pointer (class)
bad_weak_ptr C++11	Bad weak pointer (class)
ios_base::failure	Base class for stream exceptions (public member class)
logic_error	Logic error exception (class)
runtime_error	Runtime error exception (class)

http://www.cplusplus.com/reference/exception/exception/

```
#include <iostream>
#include <exception>

int main() {
  int negative = -1;

  try {
    new int[negative];
  }
```

```
  catch(std::exception &e) {
    std::cout << e.what() << '\n'; //what is a function in the exception class
  }

  std::cout<<"Exiting..."<<std::endl;
}
```

Creating your own exception class:

Create a program where users can enter prices into grocery store items. A price higher than 100 dollars is considered too high. Create your own custom exception to handle this.

```
computer$ g++ practice.cpp
computer$ ./a.out
Enter item name and price:
Water 900
This price is ridiculously high.  No one will buy it.
Unacceptable price that was entered: $900
```

We will basically "wrap up" up exception info (encapsulation) into an exception object and throw that object. Before we were throwing ints, strings etc, but now we are throwing whole exception objects.

```
#include <iostream>
#include <string>
#include <vector>
#include <exception> // for std::exception

using namespace std;

//create your own custom exception class
class price_exception: public exception{
 int price;
public:

  const char* what() //override the what() function in the exception class.  This is the function signature in the exception class (notice it is virtual: virtual const char* what() const noexcept;)
```

If you are interested in const functions (meaning it won't modify the object it is dealing with):
https://www.geeksforgeeks.org/const-member-functions-c/

if you are interested more in noexcept (noting whether or not the function can throw an exception):
https://en.cppreference.com/w/cpp/language/noexcept_spec

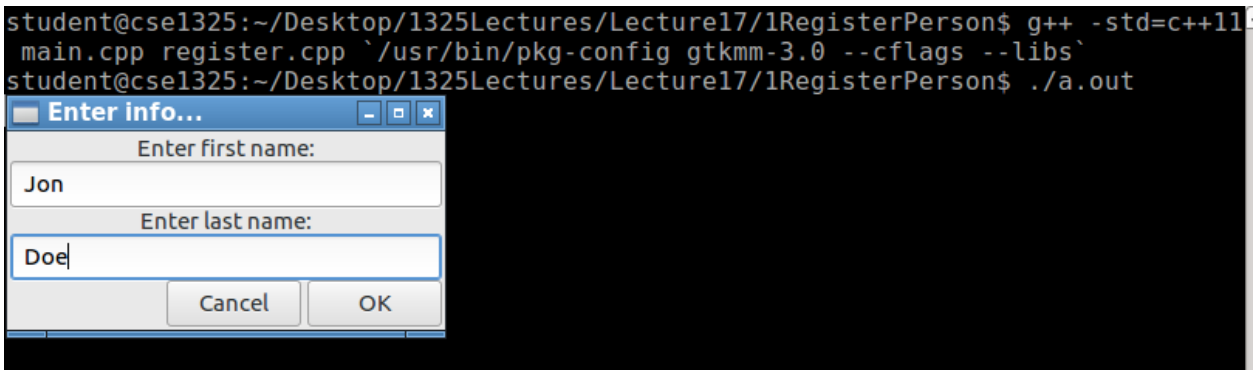
```
 {
     return "This price is ridiculously high.  No one will buy it.";
 }

 void setprice(int price)
 {
```

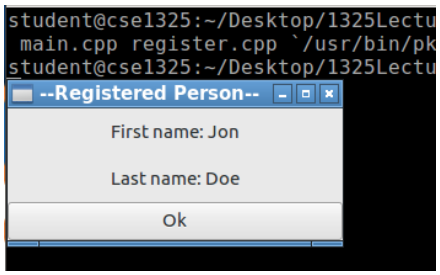
```
      this->price=price;
  }

  void getprice()
  {
       cout<<"Unacceptable price that was entered: $"<<price<<endl;
  }
};

class grocery_item{
  int price;
  string name;
public:
  friend class Wholefoods; //friend class so Wholefoods class can access private members
};


class Wholefoods{
  vector<grocery_item> all_items;

public:
  void add_item()
  {
            grocery_item i;
            cout<<"Enter item name and price:"<<endl;
            cin>>i.name>>i.price;

           if(i.price>100)
           {
                price_exception p; //create custom exception object
                p.setprice(i.price); //include the price that caused the exception
                throw p; //throw exception
           }
    }
};

int main(int argc, char**argv)
{
  Wholefoods store1;

  try
  {
            store1.add_item();
  }

  catch(price_exception p)
  {
            cout<<p.what()<<endl;
            p.getprice();
  }
```

}

Program 1:

Create a program that allows a user to register using a first and last name. The name can then be shown to the user.





register.h

```
#ifndef REGISTER_H
#define REGISTER_H

#include <gtkmm.h>


class Person{

public:
  std::string first_name;
  std::string last_name;

  Person();
};



class Info_box:public Gtk::Window{

public:
```

```
  Info_box(std::string first_name, std::string last_name);

  virtual ~Info_box();

  Gtk::Label label, label1;
  Gtk::Button ok_button;
  Gtk::Grid grid;
  Gtk::VBox vbox;
  Gtk::HBox hbox;

protected:
  void ok_function();

};

#endif
```

register.cpp

```
#include <gtkmm.h>
#include "register.h"
#include <iostream>
#include <string>
#include <vector>


  Person::Person()
  {
                Gtk::Window w;   //you will get a warning if you don't include this
                Gtk::Dialog *dialog = new Gtk::Dialog();
                dialog->set_transient_for(w); //you will get a warning if you don't include this
                dialog->set_title("Enter info...");

                Gtk::Label *label = new Gtk::Label("Enter first name:");
                dialog->get_content_area()->pack_start(*label);
                label->show();

                dialog->add_button("Cancel", 0); //Creating a button called "Cancel", let 0 mean it was pressed
                dialog->add_button("OK",1);  //Creating a button called "OK", let 1 mean it was pressed

                Gtk::Entry *entry_first = new Gtk::Entry();
                entry_first->set_text("default_text");
                entry_first->set_max_length(50);
                entry_first->show();
                dialog->get_vbox()->pack_start(*entry_first);

                Gtk::Label *label1 = new Gtk::Label("Enter last name name:");
                dialog->get_content_area()->pack_start(*label1);
                label1->show();
```

```
        Gtk::Entry *entry_last = new Gtk::Entry();
        entry_last->set_text("default_text");
        entry_last->set_max_length(50);
        entry_last->show();
        dialog->get_vbox()->pack_start(*entry_last);

        int result = dialog->run(); //running the dialog window

        if(result==1) //OK button was pushed
        {
                first_name = entry_first->get_text(); //setting the first and last name of the Person object
                last_name=entry_last->get_text();
        }

        else //Cancel button was pushed-set first/last name to CANCELED
        {
                first_name = “CANCELED”;
                last_name= “CANCELED”;

        }

        dialog->close();

        delete dialog;
        delete label;
        delete entry_first;
        delete entry_last;

}


//We are passing two strings into the constructor of the box (the first and last name)
Info_box::Info_box(std::string first_name, std::string last_name)
{
    set_title("--Registered Person--");
    set_size_request(150, 100);
    add(vbox);

    label.set_text("First name: "+first_name);
    label.set_padding(10,10);
    vbox.pack_start(label);

    label1.set_text("Last name: "+last_name);
    label1.set_padding(10,10);
    vbox.pack_start(label1);

    ok_button.set_label("Ok");

    ok_button.signal_pressed().connect(sigc::mem_fun(*this,&Info_box::ok_function));
```

```
    vbox.pack_start(ok_button);
    show_all_children();
 }

 Info_box::~Info_box(){};

 void Info_box::ok_function()
 {
        hide();
 }
```

main.cpp

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

int main(int argc, char **argv)
{

  Gtk::Main app(argc, argv);
  Person p; //creating a Person object (with first/last name input)

  Info_box window(p.first_name, p.last_name); //using Person info in window constructor

  Gtk::Main::run(window);

  return 0;
}
```