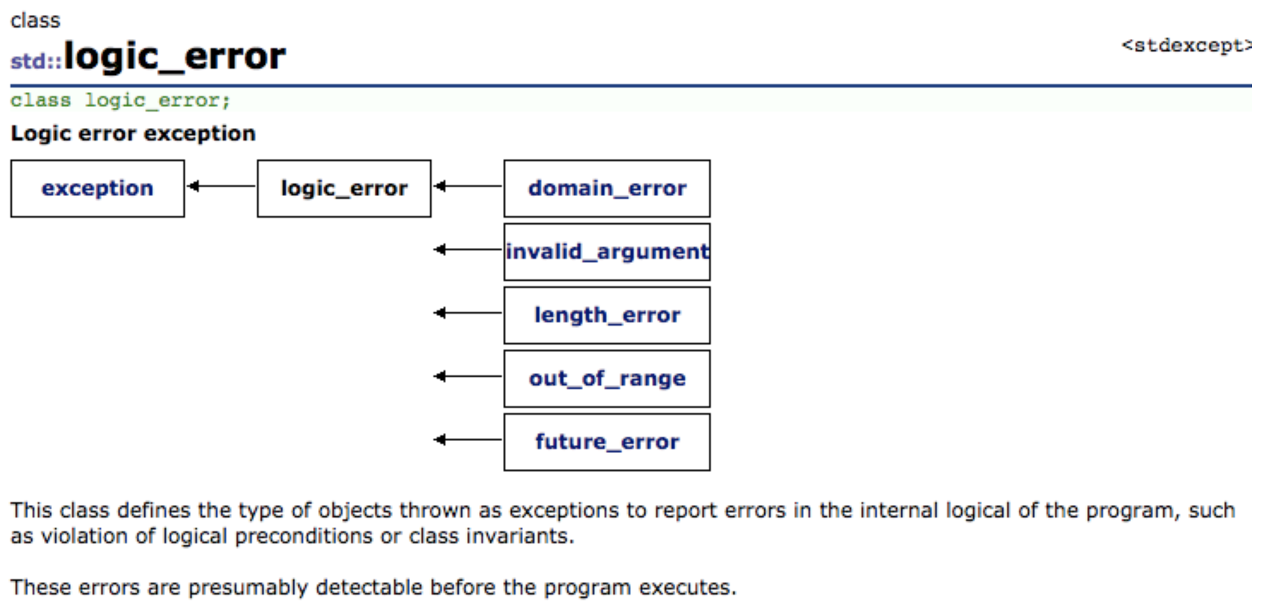
Coding Review

YOU SHOULD FREE ANY POINTERS

Problem 1:



class
std::**logic_error**   <stdexcept>

`class logic_error;`

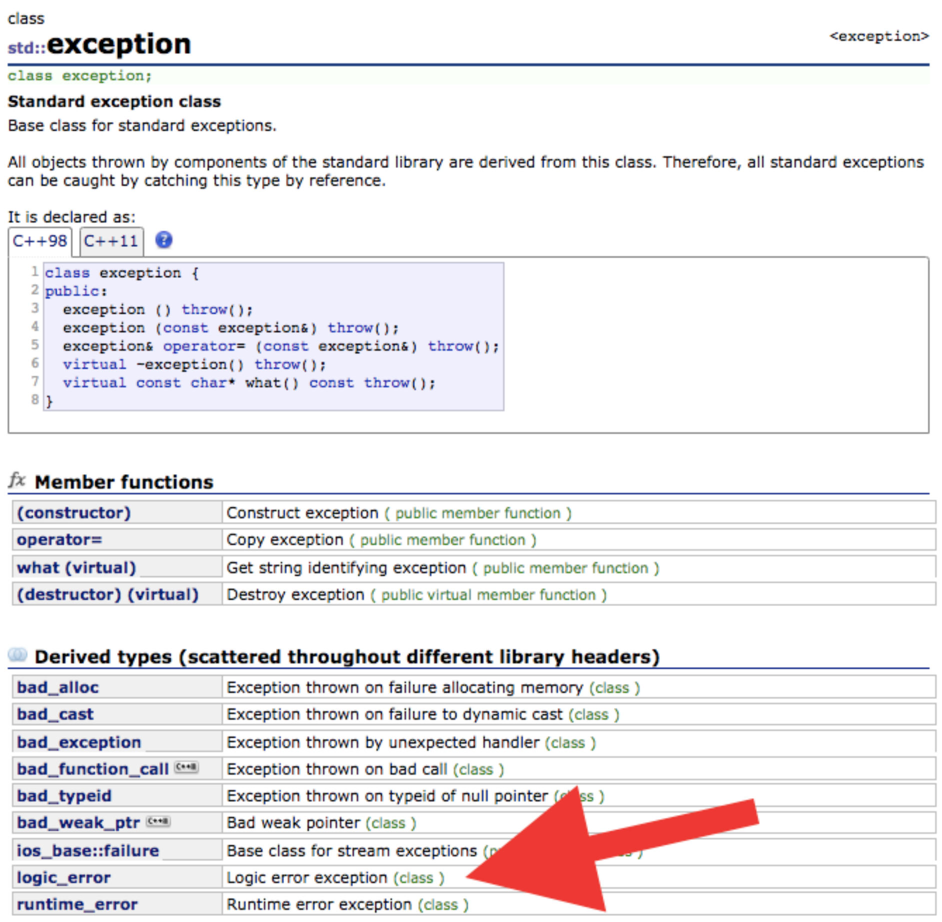
Logic error exception

This class defines the type of objects thrown as exceptions to report errors in the internal logical of the program, such as violation of logical preconditions or class invariants.

These errors are presumably detectable before the program executes.

http://www.cplusplus.com/reference/stdexcept/logic_error/

out_of_range is a class from *logic_error* (which is a class from *exception*):



class
std::**exception**   <exception>

`class exception;`

Standard exception class
Base class for standard exceptions.

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.

It is declared as:
C++98 | C++11

```
class exception {
public:
  exception () throw();
  exception (const exception&) throw();
  exception& operator= (const exception&) throw();
  virtual ~exception() throw();
  virtual const char* what() const throw();
}
```

Member functions

(constructor)	Construct exception (public member function)
operator=	Copy exception (public member function)
what (virtual)	Get string identifying exception (public member function)
(destructor) (virtual)	Destroy exception (public virtual member function)

Derived types (scattered throughout different library headers)

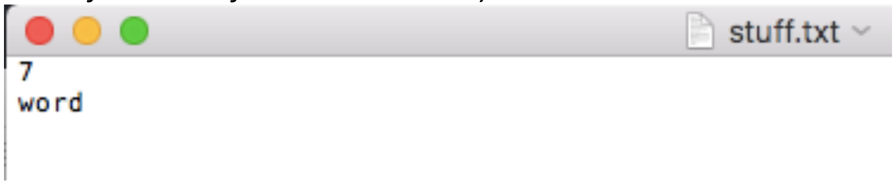
bad_alloc	Exception thrown on failure allocating memory (class)
bad_cast	Exception thrown on failure to dynamic cast (class)
bad_exception	Exception thrown by unexpected handler (class)
bad_function_call C++11	Exception thrown on bad call (class)
bad_typeid	Exception thrown on typeid of null pointer (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/

Using a template, create a stack.

```
computer$ g++ practice.cpp
computer$ ./a.out > stuff.txt
Exception: Empty stack when trying to pop element.
```

Note that I am using redirection here and the print statement goes to the screen because I am using cerr (see below for more info with cerr vs cout)



```
7
word
```

Topics covered: Templates, exception handling, error stream

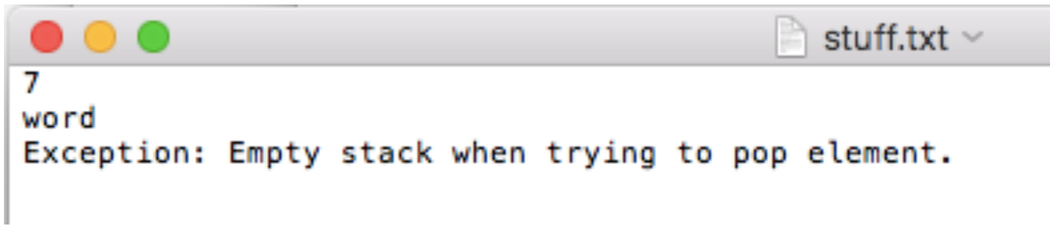
```
#include <iostream>
#include <vector>
#include <string>
#include <exception>

using namespace std;

template <class T>
class Stack {
  private:
    vector<T> elems;    // elements in stack

  public:

    // push element
    void push(T elem)
 {
 elems.push_back(elem);
 }

    // pop element (syntax to define function outside class-see below)
    void pop();

    // return top element (syntax to define function outside class-see below)
    T top();

    // return true if empty (syntax to define function inside class)
    bool empty()
    {
      return elems.empty();  // return true if empty.
    }
};
```

```
template <class T>
void Stack<T>::pop ()
{
  if (elems.empty())
  {
    throw out_of_range("Empty stack when trying to pop element.");
  }
  elems.pop_back();
}

template <class T>
T Stack<T>::top ()
{
  if (elems.empty())
  {
    throw out_of_range("Empty stack when trying to see top element.");
  }

  // return copy of last element
  return elems.back();
}

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

 try {
    Stack<int>  stack_example_int;  //create a stack of ints
    Stack<string> stack_example_string;    //create stack of strings

    //using int stack
    stack_example_int.push(7);
    cout << stack_example_int.top() <<endl;

    //using string stack
    stack_example_string.push("word");
    cout << stack_example_string.top() << std::endl;
    stack_example_string.pop();
    stack_example_string.pop();
  }
  catch (exception const& ex)
  {
    cerr << "Exception: " << ex.what() <<endl;
  }
}
```

<u>Notes on the code:</u>

1. Notice I have the possibility to throw two exceptions above (they would end up in the catch block if the thrown)
2. By using *cerr* instead of *cout*, we have the following difference when running the program (notice the sample run below does not print to screen the print statement when using *cout instead of cerr*- this is one of the differences between the two).

```
computer$ g++ practice.cpp
computer$ ./a.out >stuff.txt
computer$
```



```
7
word
Exception: Empty stack when trying to pop element.
```

Note that vector (what we have been using from the STL library-Standard **Template** Library) is declared similarly to we did above:

template < class T, class Alloc = allocator<T> > class vector;

Thea additional parameter (allocator) is also a template. It is used to dynamically handle storage for the vector.

template <class T> class allocator;

Problem 2:

Topics covered: iterators, maps, enum classes, friend functions, static casting

```
computer$ g++ -std=c++14 practice.cpp
computer$ ./a.out
Watercooler has 2 bottle(s)!
b1 is full!

b1      value: 0
b2      value: 2
```

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

using namespace std;

enum class bottlestatus {FULL,HALF_FULL,EMPTY};

class Watercooler{

  map<string,bottlestatus> all_bottles; //we need to specify our value as the enum type

public:
  void add_bottle(string name, bottlestatus status)
  {
```

```
  all_bottles.insert({name,status});
  }

  void show_info()
  {

    for  (map<std::string, bottlestatus>::iterator it=all_bottles.begin(); it!=all_bottles.end(); it++)
    {
             cout << it->first << " value: " << static_cast<int>(it->second) << endl; // notice we have to static cast to print out as an integer because we are using an enum class (see below when not using an enum class)
     }
  }

  friend ostream& operator <<(ostream& os, Watercooler w);
};

ostream& operator<<(ostream& os, Watercooler w)
{
  os<<"Watercooler has "<<w.all_bottles.size()<<" bottle(s)!"<<endl;

  if(w.all_bottles.at("b1")==bottlestatus::FULL)
  {
       os<<"b1 is full!"<<endl;
  }

  else
  {
      os<<"b1 is NOT full!"<<endl;
  }

  return os;
}

int main(int argc, char**argv)
{
  Watercooler w1;
  w1.add_bottle("b1", bottlestatus::FULL);
  w1.add_bottle("b2", bottlestatus::EMPTY);

  cout<<w1<<endl;
  w1. show_info();


}
```

Notice when we DON'T use an enum class (just a normal enum):

```
computer$ ./a.out
b1       value: 0
```

```
b2     value: 2
```

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

using namespace std;

enum bottlestatus {FULL,HALF_FULL,EMPTY};

class Watercooler{

  map<string,int> all_bottles; //notice we can make the value an int (when using a class, we need to
mention the enum class name)

public:
  void add_bottle(string name, int status)
  {
      all_bottles.insert({name,status});
  }

  void show_info()
  {
    for (map<std::string, int>::iterator it=all_bottles.begin(); it!=all_bottles.end(); it++)
    {
       cout << it->first << " value: " << it->second << endl; //notice we do not have to static cast here (the
enum is already acting as an integer)
    }
  }
};

int main(int argc, char**argv)
{
  Watercooler w1;
  w1.add_bottle("b1", FULL);
  w1.add_bottle("b2", EMPTY);

  w1.show_info();

}
```

Note about iterators:

//Notice when declaring our iterator we say: map::iterator. Iterator is a type inside our map class so we access it using ::

//begin() returns an iterator to the first element, end() returns an iterator to the last element

```
void show_info()
 {
   for (map<std::string, int>::iterator it=all_bottles.begin(); it!=all_bottles.end(); it++)
   {
      cout << it->first << " value: " << it->second << endl;
   }
 }
```

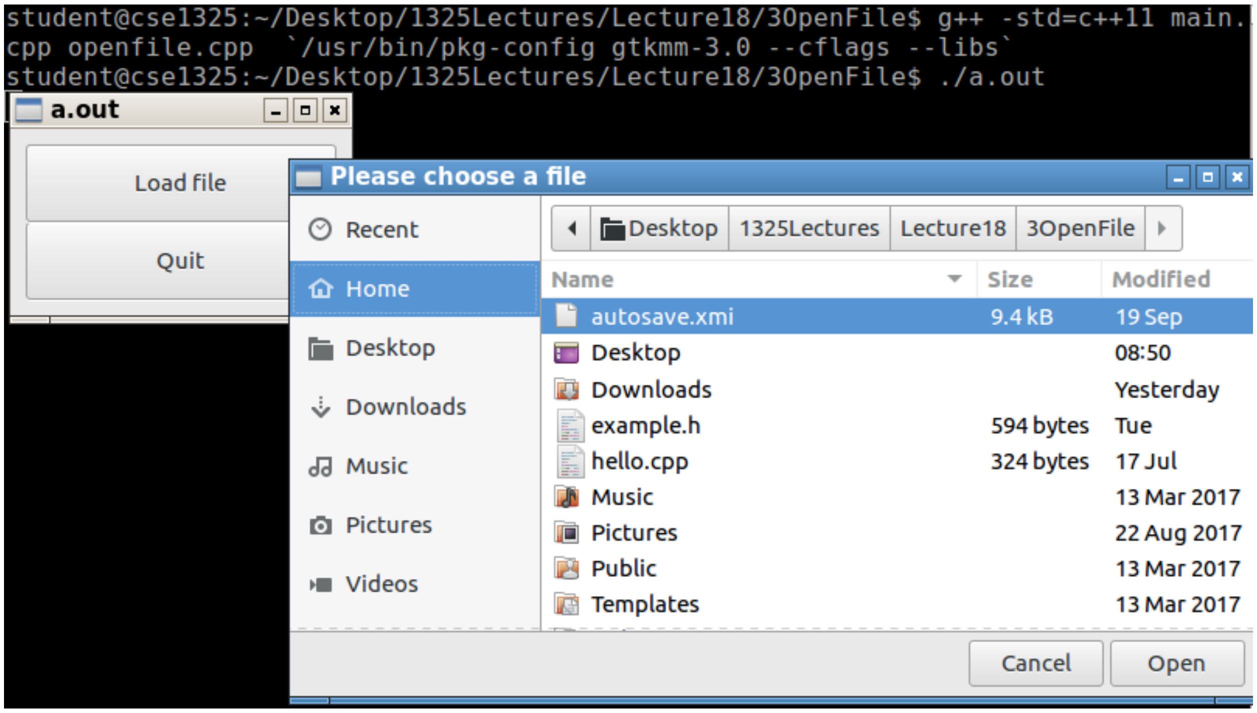
//first is the key, second is the value

//Note: our map class has a type called value_type. This is defined as a pair (pair is a class template). First accesses the key and second accesses the value. Our iterator is pointing at a pair, so when we dereference, we need to specify which in the pair we are looking at.

//See following websites for more info:
https://www.tutorialspoint.com/cpp_standard_library/map.htm
http://www.cplusplus.com/reference/utility/pair/pair/
http://www.cplusplus.com/reference/iterator/iterator/
http://www.cplusplus.com/reference/iterator/BidirectionalIterator/

Problem 3:

Allow a user to choose a file.



```
student@cse1325:~/Desktop/1325Lectures/Lecture18/3OpenFile$ g++ -std=c++11 main.cpp openfile.cpp  `/usr/bin/pkg-config gtkmm-3.0 --cflags --libs`
student@cse1325:~/Desktop/1325Lectures/Lecture18/3OpenFile$ ./a.out
```

Name	Size	Modified
autosave.xmi	9.4 kB	19 Sep
Desktop		08:50
Downloads		Yesterday
example.h	594 bytes	Tue
hello.cpp	324 bytes	17 Jul
Music		13 Mar 2017
Pictures		22 Aug 2017
Public		13 Mar 2017
Templates		13 Mar 2017

openfile.h

```
#ifndef File_window_H
#define File_window_H

#include <gtkmm.h>

class File_window : public Gtk::Window
```

```
{

public:
  File_window();
  virtual                    ~File_window();

protected:
  void                        on_button_load();

  //widgets:
  Gtk::Button            openfile;
  Gtk::Button            quit;
  Gtk::Grid                grid;

};


#endif
```

openfile.cpp

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


File_window::File_window()
{
  this->set_border_width(10);

  openfile.add_label("Load file");
  openfile.set_size_request(200,50);
  openfile.signal_pressed().connect(sigc::mem_fun(*this,&File_window::on_button_load));
  grid.attach(openfile,0,0,1,1);

  quit.add_label("Quit");
  quit.set_size_request(200,50);
  quit.signal_pressed().connect(sigc::mem_fun(*this,&File_window::close));
  grid.attach(quit,0,1,1,1);

  grid.show_all();

  add(grid);
}


File_window::~File_window()
{}


void File_window::on_button_load()
{
  //This creates the file chooser dialog that pops on screen
  Gtk::FileChooserDialog dialog("Please choose a file",Gtk::FILE_CHOOSER_ACTION_OPEN);
  dialog.set_transient_for(*this); //you will get a warning if you don't include this

  dialog.add_button("_Cancel", Gtk::RESPONSE_CANCEL);
  dialog.add_button("_Open", Gtk::RESPONSE_OK);

  int result = dialog.run(); //(like the first program)-the return value will be what button they pushed.
We have two options shown above: RESPONSE_CANCEL or RESPONSE_OK.  Remember these are enums
(you can look them up) and they correspond to values.  Note we could have just assigned any number
(like I did in the first program)

  //use the result from above
  switch(result)
  {
  case(Gtk::RESPONSE_OK): //open file
  {
      std::cout << "Open." << std::endl;
      std::string filename = dialog.get_filename();
      std::cout << "File:" <<  filename << std::endl;
      break;
  }
  case(Gtk::RESPONSE_CANCEL): //cancel
  {
     std::cout << "Cancel." << std::endl;
     break;
  }
  default:
  {
     std::cout << "!!!" << std::endl;
     break;
  }
  }
}
```

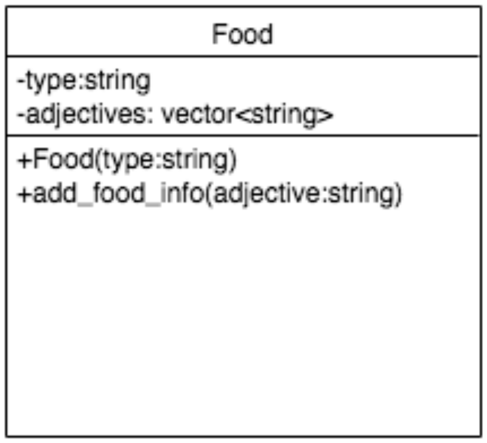
main.cpp

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


int main(int argc, char* argv[])
{
 Gtk::Main app(argc, argv);
 File_window w;
 Gtk::Main::run(w);
 return 0;
}
```

Problem 4:

A Food object has a type (the name of the food) and a list of up to three adjectives. Any food can take any adjective except cucumbers (they can take any adjective except *cool*).

1. Create a food class using the given class diagram



2. Create a custom exception (it should be a class) that is thrown when a user tries to enter the adjective *cool* with a cucumber food object (when the function *add_food_info* is used)

3. Create a program (using the Food class and custom exception you created) to match the sample run

Sample run:

```
Enter food type:
cucumber

--Enter adjective:
green

--Enter adjective:
cool
No cool cucumbers accepted.
cucumber and cool not allowed together.

--Enter adjective:
cheap

--Enter adjective:
cool
No cool cucumbers accepted.
cucumber and cool not allowed together.

--Enter adjective:
seedy
```

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

using namespace std;

class cucumber_exception:public exception{
  string type;
  string adjective;

public:
  const char* what()
  {
    return "No cool cucumbers accepted.";
  }

  void set_info(string type, string adjective)
  {
    this->type=type;
    this->adjective=adjective;
  }

  void print_info()
  {
    cout<<type<<" and "<<adjective<<" not allowed together."<<endl;
  }
};

class Food{
  string type;
  vector<string> adjectives;

public:
   Food(string type)
  {
    this->type=type;
  }

  void add_food_info(string adjective)
  {
    if(type=="cucumber" && adjective=="cool")
    {
      cucumber_exception c1;
      c1.set_info(type, adjective);
      throw c1;
    }

    adjectives.push_back(adjective);
  }
};


int main(int argc, char **argv)
{
  string answer;
  cout<<"Enter food type:"<<endl;
  cin>>answer;

  Food f1(answer);

  for(int i=0;i<3;i++) //can also use a while loop
  {
    cout<<"\n--Enter adjective: "<<endl;
    cin>>answer;
    try
    {
      f1.add_food_info(answer);
    }
    catch(cucumber_exception e)
    {
      cout<<e.what()<<endl;
      e.print_info();
      i--;
    }

  }

}
```

Problem 5:

An airplane can be identified by either an id number (given as an integer) or a string. Every airplane has a specific destination it flies to (which is changeable with the overloaded operator <<). An airplane can fly to any destination other than Antarctica.

Complete the program by creating a class that would work with the following main and the given sample run.

Sample Run:

```
33: Changing destination from New York to Morocco
id_plane123: Won't fly to Antarctica...
```

```
int main(int argc, char **argv)
{
    stuff::Airplane <int> a1("New York", 33);
    stuff::Airplane <string> a2("Las Vegas", "id_plane123");
    a1<<"Morocco";
```

```
      a2<<"Antarctica";

   }
```

Possible solution:

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

using namespace std;

namespace stuff{

  template <class T>
  class Airplane{
    T pass_id;
    string destination;
    public:

    Airplane(string dest,T pass_info)
    {
        pass_id=pass_info;
        destination=dest;
    }

     void operator <<(string s) //overloaded operator
     {
        if(s!="Antartica")
        {
          cout<<pass_id<<": Changing destination from "<<destination<< " to "<<s<<endl;
          destination=s;
        }

        else
        {
          cout<<pass_id<<": Won't fly to Antartica..."<<endl;
        }

     }

  };
}
```

Problem 6:

A bank account is considered to be good (see sample run-first thread output) if it has more than 50 dollars. A bank account is considered not to be good if it is 50 dollars or less (see sample run-second thread output).

Complete the given main and the function *verify_info* below to match the sample run below. The function takes three parameters. The *name* parameter is the name of the person checking the bank account. The *total* parameter is the amount of money in the bank account. The third parameter is a mutex.

```
void verify_info(string name, int total, std::mutex &mtx);

int main(int argc, char **argv)
{
   ___________________________ //create a mutex object

   ___________________________ //create first thread
   ___________________________ //create second thread

  ___________________________ //join first thread
  ___________________________ //join second thread

}
```

Sample run:

```
//First thread:
Checking...please wait... //there should be a pause of 2 seconds here
You're good Bobby!
//Second thread:
Checking...please wait... //there should be a pause of 2 seconds here
Running low on cash!
```

Possible solution:

```
#include<iostream>
#include <string>
#include <thread>
#include <mutex>
#include <chrono>

using namespace std;


//below budget
void verify_info(string name, int total, std::mutex &mtx)
{
  std::lock_guard<std::mutex>lock(mtx);
  cout<<"Checking...please wait..."<<endl;
  std::this_thread::sleep_for(std::chrono::seconds(2));

  if(total<50)
  {
    cout<<"Running low on cash!"<<endl;
  }
  else
  {
    cout<<"You're good "<<name<<"!"<<endl;
  }
}


int main(int argc, char** argv)
{
  std::mutex mtx;

  std::thread t1(verify_info,"Bobby", 60, ref(mtx));
  std::thread t2(verify_info,"Jane",45,ref(mtx));

  t1.join();
  t2.join();

}
```

Problem 7:

1. Given the following classes, create a class Diagram for each class

```
class Potato{

  protected:
    float price;

  public:
    Potato(float price)
   {
      this->price=price;
   }

   virtual void change_price(float percentage)=0;
};


class Russet_potato:public Potato{

  public:
    using::Potato::Potato;

    void change_price(float percentage)
    {
      price=price* (1+percentage);
      cout<<"New price is: $"<<price<<endl;
    }

    void add_butter()
   {
     cout<<"Adding butter!"<<endl;
   }
};

class Red_potato:public Potato{
  int size;

  public:
  Red_potato(int size, float price):Potato(price)
  {
    this->size=size;
  }

  void chop()
  {
    cout<<"Chop!"<<endl;
  }

  void change_price(float percentage)
  {
    price=price-(price*(percentage));
    cout<<"New price is: $"<<price<<endl;
  }

};
```

2. Using these classes, create a complete program by defining a main to match the sample run

<u>Sample Run:</u>

```
How many potatoes?
3
Enter potato type (Russet or Red):
Red
Enter potato type (Russet or Red):
Russet
Enter potato type (Russet or Red):
Russet

1: Enter percentage to change to (for Red potato):
.3
New price is: $2.793
Chop!

2:Enter percentage to change to (for Russet Potato):
.4
New price is: $4.186
Adding butter!

3:Enter percentage to change to (for Russet Potato):
.2
```

```
New price is: $3.588
Adding butter!
```

Possible main:

```
 int main(int argc, char**argv )
 {
    vector<Potato*> all_potato;
    int answer, i=0;
    string answer1;
    cout<<"How many potatoes?"<<endl;
    cin>>answer;
    Potato *p1;

    while(i<answer)
    {
      cout<<"Enter potato type (Russet or Red):"<<endl;
      cin>>answer1;

      if(answer1=="Russet")
      {
        p1=new Russet_potato(2.99);
      }

      else
      {
        p1=new Red_potato(1,3.99);
      }

      i++;
      all_potato.push_back(p1);
    }

    for(int i=0;i<answer;i++)
    {
      Russet_potato *r=dynamic_cast<Russet_potato*>(all_potato[i]);

      if(r==NULL)//not a russet potato
      {
        Red_potato *r1=dynamic_cast<Red_potato*>(all_potato[i]);

        float perc;
        cout<<"\n"<<i+1<<": Enter percentage to change to (for Red potato): "<<endl;
        cin>>perc;
        r1->change_price(perc);
        r1->chop();
      }

      else //is a russet potato
      {
        float perc;
        cout<<"\n"<<i+1<<":Enter percentage to change to (for Russet Potato): "<<endl;
        cin>>perc;

        r->change_price(perc);
        r->add_butter();
      }

    }

}
```