Final Examination – solutions
(Tuesday, February 18, 2004: 12:15pm – 15:15pm)
(Closed Book, Closed Notes)

“I certify that I have neither received nor given un-permitted aid on this examination and that I have reported all such incidents observed by me in which un-permitted aid is given.”

Signature __________ SOLUTIONS __________
Name _______________________________
Student ID ___________________________

Question 01 -------------- / [6]
Question 02 -------------- / [6]
Question 03 -------------- / [8]
Question 04 -------------- / [18]
Question 05 -------------- / [14]
Question 06 -------------- / [8]
Question 07 -------------- / [8]
Question 08 -------------- / [8]
Question 09 -------------- / [11]
Question 10 -------------- / [8]
Question 11 -------------- / [2]
Question 12 -------------- / [2]
Question 13 -------------- / [0]
Question 14 -------------- / [1]

===================

Total: ____________/100
Q 1 [6 pts]: Consider the following code which depicts a multiple inheritance scenario:

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

class RockMusic
{
protected:
    string treble;
    string bass;
public:
    RockMusic(string t, string b):treble(t), bass(b){}
    void electricGuitar() {cout<<"Electric guitars rock!"<<endl;}
    void famous(){cout<<"Guns n' Roses"<<endl;}
    void drumming(){cout<<"Loud and Fast"<<endl;}
};

class Rap
{
protected:
    string lyrics;
public:
    Rap(string l):lyrics(l){}
    void famous(){cout<<"Eminem"<<endl;}
    void Rhythm(){cout<<"Good rhythm"<<endl;}
};

class AlternativeMusic:public Rap, public RockMusic
{
private:
    string band;
public:
    AlternativeMusic(string l, string t, string b, string n):
        Rap(l), RockMusic(t, b), band(n){}
};

Now consider the `main()` below. Fix all the errors in this main, if any. If there are multiple ways of fixing errors, use one that pleases you. Indicate the output after fixing.

```cpp
int main()
{
    AlternativeMusic group1("good", "reasonable treble", "good bass", "Linkin Park");
    group1.electricGuitar();
    group1.Rhythm();
    group1.Rap::famous();
    AlternativeMusic group2("good", "none", "okay bass", "EP");
    group2.electricGuitar();
    group2.Rhythm();
    group2.drumming();
    return 0;
}
```

Correction shown in **BOLD**

The output will be:

```
Electric guitars rock!
Good rhythm
Eminem
Electric guitars rock!
Good rhythm
Loud and Fast
```
Q 2 [6 pts]: Consider assigning the return value from a function (or a method) to a reference. There are two possibilities:

(i) Return value from a global function is assigned to an object reference like this:

\[
\text{someClass} & \text{ someObject} = \text{foo();}
\]

(ii) Return value from a method call on an object is assigned to an object reference:

\[
\text{someClass} & \text{ someObject} = \text{someOtherObject.bar();}
\]

Both functions \text{foo()} and \text{bar()} return an object either by reference or by value. Without knowing and assuming anything about the return types of \text{foo()} and \text{bar()}, answer the following:

(a) [3 pts] Is it safe to use the statement in (i) Yes/No? No

If yes, then why?

If not, why not?

If the function is returning by value, we will create a reference to an unnamed temporary. Thus, without knowing the return type, it is unsafe to use the given statement. Were the function returning a reference, it would be a reference to an object which will go out of scope because the function is taking no parameters.

(b) [3 pts] Is it safe to use the statement in (ii)? Yes/No? No

If yes, then why?

If not, why not?

If the function returns a local object by value or by reference, we will be left with a reference to an unnamed temporary which is not okay. In fact, compiler will not let us do that. However, it is okay if an encapsulated object (data member of the class) is returned either by reference or by value. Since we do not know the implementation of called method, it is not safe to use it the way it is given.
Q 3 [8 pts]:
(a) [4 pts] Consider the following code that uses inheritance:

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

class A
{
public:
    virtual void Foo()
    {
        cout << "Foo() of class A" << endl;
        Bar(-5);
    }

    void Bar(int num)
    {
        cout << "Bar(int) of class A where int is: " << num << endl;
    }
};

class B : public A
{
public:
    void Foo()
    {
        cout << "Foo() of class B" << endl;
        Bar(-10);
    }

    void Bar(int num)
    {
        cout << "Bar(int) of class B where int is: " << num << endl;
    }
};
class C : public B
{
public:
    void Bar(int num)
    {
        cout << "Bar(int) of class C where int is: " << num << endl;
    }
};

void main()
{
    A* a = new B;
    a->Foo();
    a->Bar(23);
    B* b = new C;
    b->Foo();
    b->Bar(96);
}
```

For the main() shown below, indicate the output in the space provided.

```
Foo() of class B
Bar(int) of class B where int is: -10
Bar(int) of class A where int is: 23
Foo() of class B
Bar(int) of class B where int is: -10
Bar(int) of class B where int is: 96
```
(b) [4 pts] The following code is similar to the one in the previous part except that the virtual-ness of class methods is altered (carefully read through the function prototypes):

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

class A
{
    public:
        void Foo()
        {
            cout << "Foo() of class A" << endl;
            Bar(-5);
        }

        virtual void Bar(int num)
        {
            cout << "Bar(int) of class A where int is: " << num << endl;
        }
};

class B : public A
{
    public:
        void Foo()
        {
            cout << "Foo() of class B" << endl;
            Bar(-10);
        }

        void Bar(int num)
        {
            cout << "Bar(int) of class B where int is: " << num << endl;
        }
};

class C : public B
{
    public:
        void Bar(int num)
        {
            cout << "Bar(int) of class C where int is: " << num << endl;
        }
};
```

For the main() shown below, indicate the output in the space provided.

```cpp
void main()
{
    B* b = new C;
    FooBar(b);
}
```

<table>
<thead>
<tr>
<th>Void FooBar(A* a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-&gt;Foo();</td>
</tr>
<tr>
<td>(dynamic_cast&lt;C*&gt;(a))-&gt;Bar(99);</td>
</tr>
<tr>
<td>(dynamic_cast&lt;C*&gt;(a))-&gt;Foo();</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Foo() of class A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar(int) of class C where int is: -5</td>
</tr>
<tr>
<td>Bar(int) of class C where int is: 99</td>
</tr>
<tr>
<td>Foo() of class B</td>
</tr>
<tr>
<td>Bar(int) of class C where int is: -10</td>
</tr>
</tbody>
</table>

For the main() shown below, indicate the output in the space provided.

```cpp
Foo() of class A
Bar(int) of class C where int is: -5
Bar(int) of class C where int is: 99
Foo() of class B
Bar(int) of class C where int is: -10
```
Q 4 [18 pts]: In this problem, you are required to write an Employee class.

The class is required to have the following data members, at the least:

- A C++ string to hold the name of the employee
- An Employee** to hold the coworkers (see details below)
- An int to hold the current number of coworkers of the employee
- An int to hold the maximum number of coworkers of the employee

The class is also required to implement the following functionality, at the least:

- On construction, all the data members should be in “known” states.
- Each employee must be constructed with a name and the maximum number of coworkers. You must provide a mechanism such that it is impossible to construct an employee if name or maximum number of coworkers is missing.
- Each Employee object will own an array of Employee pointers, allocated on the heap during construction. Therefore, objects must store a pointer to that array.
- A setter for maximum number of coworkers.
- A method bool addCoworker(Employee&) to add a coworker. When you make Y a coworker of X by calling this method on X, this method should automatically make X a worker of Y. The method will only succeed and return true if the current number of coworkers of both X and Y are less than their respective maximum numbers of coworkers and these two employees are not coworkers previously. You must not add Y to the coworker list of X or vice versa within this function if this function is going to return false.
- A void print() method which prints name of employee, number of coworkers, and names of all coworkers on a new line.
- A special method void setMaxCoworkers(int) which must do something appropriate with the internal allocation of the array of Employee pointers.
- You might also want to provide a destructor for this class.
- You are REQUIRED not to add anything else to the public interface!

Conforming to above requirements, here is the partial class definition:

```cpp
class Employee
{
private:
    string name;
    Employee** coworkers;
    int numCoworkers;
    int maxCoworkers;
    // add other private methods and data members here
public:
    // add a constructor if you wish
    ~Employee();
    bool addCoworker(Employee&);
    void print();
    void setMaxCoworkers(int);
    // do not add any other public methods
};
```

You are required to provide the complete class definition and implementation of each function. To assist you in code writing, we are providing a sample main() with some comments.
A sample main function with comments:

```c++
int main()
{
    Employee e1("Babar", 2); // declares an Employee object e1
    Employee e2("Naheed", 3); // e2 can have at most 3 coworkers
    Employee e3("Arif", 1); // e3 can have at most one coworker
    Employee e4("Binish", 1);

    e1.addCoworker(e2); // Adds e1 as coworker of e2 and vice versa
    e1.addCoworker(e2); /* returns false (e1 and e2 are already coworkers)
                              state of e1 and e2 remained unchanged */
    e2.addCoworker(e3); // returns true
    e3.addCoworker(e1); /* returns false because e3 can’t have any more
                              state of e3 and e1 remained unchanged */
    e4.addCoworker(e3); /* returns false because e3 can’t have any more
                              state of e3 and e4 remained unchanged */

    e1.print(); // prints names of e1 and its one coworker (e2)
    e2.print(); // prints names of e2 and its two coworkers (e1, e3)
    e3.print(); // prints names of e3 and its one coworker (e2)
    e4.print(); // prints name of e4 who has no coworkers

    e3.setMaxCoworkers(5); // now e3 can have a max of 5 coworkers
    e4.addCoworker(e3); // make e3 and e4 coworkers

    e3.print(); // prints names of e3 and its two coworkers (e2, e4)
    e4.print(); // prints names of e4 and its one coworkers (e3)

    return 0;
}
```

(a) [6 pts] Provide class definition and implementation of all methods:

// write here whatever will go in your Employee.h file

One sample solution is given below:

```c++
class Employee
{
private:
    string name;
    Employee** coworkers; // pointer to array of Employee pointers
    int numCo; // current number of coworkers
    int maxCo; // maximum number of coworkers
    bool canAddCoworker() const {return numCo != maxCo;}
    void addToList(Employee& toAdd) {coworkers[numCo++] = &toAdd;}
    bool alreadyCoworker(Employee& other);

public:
    Employee(string name, int maxCo):name(name), maxCo(maxCo) {numCo=0;
        coworkers= new Employee*[maxCo];}
    ~Employee() {delete [] coworkers;}
    string getName() const {return name;}
    bool addCoworker(Employee& other);
    void setMaxCoworkers(int n);
    bool print();
};
```
Provide implementation of methods here (this will go in Employee.cc file). Include all headers that may be required.

Implementation of above sample class is given below:

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

using namespace std;

bool Employee::alreadyCoworker(Employee& other)
{
    for(int i=0; i<numCo; ++i)
    if(coworkers[i] == &other)
        return true;

    return false;
}

bool Employee::addCoworker(Employee& other)
{
    if(alreadyCoworker(other))
        return false;

    if(canAddCoworker() && other.canAddCoworker())
    {
        addToList(other);
        other.addToList(*this);
        return true;
    }

    return false;
}

void Employee::setMaxCoworkers(int n)
{
    if((n<=numCo) || (n==maxCo))
        return;

    maxCo = n;
    Employee** temp = new Employee*[n];

    for(int i=0; i<numCo; i++)
        temp[i] = coworkers[i];

    delete [] coworkers;
    coworkers = temp;
}

bool Employee::print()
{
    cout << name << "'s coworkers (current/maximum): ";
    cout << numCo << "/" << maxCo << endl;
    for (int i=0; i<numCo; ++i)
        cout << coworkers[i]->getName() << endl;
    cout << "=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=++=+=+=+=" << endl;
}
```
(b) [2 pts]: We tried storing the Employee class objects in an STL vector using the
default constructor of vector as follows:

```cpp
class Employee { ... }
vector<Employee> v1;
```

and the compiler was happy (it does not mean logic was also okay!). We then decided to
use a non-default constructor of vector to store our employees as in the following:

```cpp
class Employee { ... }
vector<Employee> v2(5);
```

and the compiler complained with a compile time error. What is going on?

In the first case, a vector is only declared while in the second case a vector of size 5 is
created and at each of those five locations, a default-constructed Employee object is
placed. Thus, the second statement calls a no-argument constructor for Employee
which is not available because we have defined our own constructor.

(c) [4 pts]: We decided to write a container class (call it MyContainer) which may hold a
bunch of Employees. Predicting that we will need MyContainer for other type of objects
as well, we decided to write it as a template.

A partial definition of our template class is given below:

```cpp
template <class T>
class MyContainer { ...
    void insert(T& item)
    {
        ...  
        localBuffer[someIndex] = item;
        ...
    }
    ...
    T getItemAt(int theIndex)
    {
        ...
        return localBuffer[theIndex];
    }
    ...
};
```

For the two methods to work properly, what needs to be modified in the original
Employee class? If some code needs to be added in that class, provide that code here.

For `insert()` to work properly, we need an implementation of `operator=`
within the class. For `getItemAt()` to work properly, a copy constructor is needed
unless we receive in an object reference. The implementations are given below:

```cpp
// copy constructor
Employee(const Employee& source)  
{  
    numCo = source.numCo;
    maxCo = source.maxCo;
    name = source.name;

    // Employee* deep copy is desirable!
    Employee** temp = new Employee*[source.maxCo];

    // Employee deep copy is not desired!
    for(int i=0; i<source.numCo; i++)
      temp[i] = source.coworkers[i];

    coworkers = temp;
}
```
(d) [3 pts]: Let us come back to storing employees in STL vectors by value, so that we will use the statement:

\[
\text{vector<Employee> v1;}\]

As we saw in part (b), compiler had no objection to this statement. We then wrote a main given in the following:

```cpp
int main()
{
    Employee e1("Babar", 2); // declares an Employee object e1
    Employee e2("Naheed", 3); // e2 can have at most 3 coworkers

    vector<Employee> v;
    v.push_back(e1);
    v.push_back(e2);

    e1.addCoworker(e2); // makes e1 and e2 coworkers

    e1.print(); // prints names of e1 and its one coworker (e2)
    e2.print(); // prints names of e1 and its one coworker (e1)

    for(vector<Employee>::iterator it=v.begin(); it!=v.end(); ++it)
        it->print(); // printed e1 and e2 without coworkers
}
```

For the above code, after making the objects e1 and e2 as coworkers, print() methods works when called on e1 and e2 but it does not work when called on objects stored in the vector. Why?

Once we push_back an employee in the vector by value, a new copy is made (by using either operator= or copy constructor depending upon the internal implementation of vector). Thus the objects stored in the vector are really copies of the original objects. Thus, adding coworkers to objects “outside” the vector has no effect on the copies stored “inside” the vector. As a result, print() methods works when called on e1 and e2 but it does not work when called on copies of e1 and e2 stored in the vector.
(e) [3 pts]: Continuing from part(d), we want objects to behave the same way whether they are in the vector or outside the vector. What are the possible solutions? Provide complete code for the solution you propose.

Considering the reason explained in part(d), one possible solution is to store pointers to Employee objects in the vector rather than storing the Employee objects. This will not require any change in Employee class code except that the main() code above should be changed to the following:

**Note: All the changes are shown in BOLD.**

```cpp
int main()
{
    Employee e1("Babar", 2); // declares an Employee object e1
    Employee e2("Naheed", 3); // e2 can have at most 3 coworkers

    vector<Employee*> v; // change Employee to Employee*
    v.push_back(&e1); // store address of e1 instead of e1
    v.push_back(&e2); // store address of e2 instead of e2

    e1.addCoworker(e2); // makes e1 and e2 coworkers

    e1.print(); // prints names of e1 and its one coworker (e2)
    e2.print(); // prints names of e1 and its one coworker (e1)

    for(vector<Employee*>::iterator it=v.begin(); it!=v.end(); ++it)
    {
        (*it)->print(); // prints e1 and e2 with coworkers 😊
    }
}
```
Q 5 [14 pts]: Consider a programming course offered at a University. The teaching staff is pretty good at remembering students’ names but can’t seem to memorize roll numbers. Unfortunately, many documents the staff has to deal with enlist students by their roll numbers and not by their names. We have to help the course staff by providing a program which takes in a student’s name from standard input (cin) and prints the roll number on the standard output (cout). Do not worry about typing errors, case sensitivity, or running the program in loop. The program takes a name, prints a roll number, and exits!!!

The following class definition is provided. Read it carefully paying special attention to return types and function parameters.

```cpp
// FileReader.h
// #include and #ifndef statements skipped for this file
class FileReader
{
private:
    // private data members and methods go here!
public:
    // Constructor: associates a file with the class
    FileReader(string& fileName);

    // Following function prepares the class to start
    // reading from the associated file. You must use this
    // before doing anything with the file.
    // This function returns false if failed.
    bool open();

    // Following function reads from the associated file.
    // Clients pass in a standard C++ pair where p.first is
    // usually a key and p.second is the corresponding value.
    // This function returns true every time a new pair is
    // read successfully. A false is returned in case of error
    // or if the end of file is encountered.
    bool readNextPair(pair<string, int>& p);

    // The following method always succeeds and is used to
    // disassociate a file from the class
    void close();
};
```

The above class is used to read data from a file in which student names and roll numbers appear at various places. It is the job of this class to read the next pair from the file.

(a) [7 pts] Assume that there is just one TA in the class who is given a file called oneFile.dat by the instructor. The format of oneFile.dat is as expected by FileReader class. Write a complete class (header and implementation) which reads from this one single file and stores the read data in one single encapsulated C++ STL map object. You will call this class StudentMapper. Provide some methods in this class such that you can write a main() function which reads one student’s name from the console using getline() function and prints out the corresponding roll number using a StudentMapper object. Provide code for that main() that you will write. You must make use of FileReader in StudentMapper. Assume that student names are unique and so are roll numbers. Provide all code for main.cc, StudentMapper.h, and StudentMapper.cc.
The `StudentMapper` class is given below:

```cpp
// file: StudentMapper.h
#include <string>
#include <map>
class StudentMapper
{
    private:
        map<string, int> m;
    public:
        StudentMapper() {}  
        void populate(string& fileName);
        int getRollNumber(string& student);
};

// file: StudentMapper.cc
#include "StudentMapper.h"
#include "FileReader.h"
#include <iostream>
#include <string>
#include <map>
using namespace std;
void StudentMapper::populate(string& fileName)
{
    FileReader reader(fileName);
    if(!reader.open())
    {
        cout << "Error!" << endl;
        exit(-1);
    }
    pair<string, int> p;
    while(reader.readNextPair(p))
    {
        m[p.first] = p.second; // we are guaranteed no repetition
    }
    reader.close();
}
int StudentMapper::getRollNumber(string& student)
{
    if(m.find(student) != m.end())
    {
        return m[student];
    }
    return -1;
}

// file: main.cc
#include "StudentMapper.h"
#include <iostream>
#include <string>
using namespace std;
int main()
{
    StudentMapper mapper;
    string fileName = "oneFile.dat";
    mapper.populate(fileName);

    string student;
    cout << "Enter Name of the Student: ";
    getline(cin, student);
    cout << mapper.getRollNumber(student) << endl;
    return 0;
}
```
(b) [7 pts] Now consider that there are five TAs for the course and each student is assigned to one TA. No student will be assigned to more than one TA. Each TA generates one data file for all the students assigned to him/her. Let us call those files one.dat, two.dat, three.dat, four.dat, and five.dat. These data files are in a format that is understandable to FileReader class. Now the instructor needs a consolidated record. Starting from the code of part (a), modify your StudentMapper.cc, StudentMapper.h and main.cc files such that the Instructor can enter the name of a student and the program will print out the student ID of that student before exiting.

Nothing needs to be changed in the StudentMapper.h and StudentMapper.cc files. The main() function in main.cc changes to the following with changes highlighted in bold:

```cpp
// file: main.cc
#include "StudentMapper.h"
#include <iostream>
#include <string>
using namespace std;
int main()
{
    StudentMapper mapper;
    string fileName = "oneFile.dat";
    mapper.populate(fileName);

    fileName = "two.dat";
    mapper.populate(fileName);

    fileName = "three.dat";
    mapper.populate(fileName);

    fileName = "four.dat";
    mapper.populate(fileName);

    fileName = "five.dat";
    mapper.populate(fileName);

    string student;
    cout << "Enter Name of the Student: ";
    getline(cin, student);

    cout << mapper.getRollNumber(student) << endl;
    return 0;
}
```
Q 6 [8 pts]: Consider a class whose name is B. We want to write another class D for which there are several options: make D a friend class of B, derive D from B, do both, etc. Fill in the following table by writing Y (for yes) or N (for no) in each square:

<table>
<thead>
<tr>
<th></th>
<th>Methods in D can access public members of B</th>
<th>Methods in D can access protected members of B</th>
<th>Methods in D can access private members of B</th>
</tr>
</thead>
<tbody>
<tr>
<td>class B</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>class D:public B{ ... }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class B</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>class D:protected B{ ... }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class B</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>class D:private B{ ... }</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class B</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>... friend class D; ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B and D not related. That is, there is no friendship relation and inheritance is not used either.</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
Q 7 [8 pts]: Consider three instantiable (i.e., non-abstract) classes E1, E2, and E3. These classes form an inheritance hierarchy such that E1 derives from E2 and E3 derives from E1 using public inheritance. Each of these classes reflects an exception that may be thrown by some piece of code.

Assume a function foo() that may throw several different kind of exceptions of which the three are E1, E2, and E3. The foo() function also throws an exception E7. We call the function foo() from within another function bar(). The code written in bar() function only knows how to handle the three types of exceptions E1, E2, and E3. All other exceptions must be re-thrown.

We call the bar() function from our main() function. The code inside our main() function knows how to handle E7 but does not know how to handle E1, E2, and E3. Thus, these three exceptions must be properly handled within bar() function.

A restriction on handling E1, E2, E3, and E7 is that they are handled by totally different pieces of code. We require that the appropriate piece of code be executed when one of these four exceptions is thrown. Another restriction on the complete program is that it must not cause any unhandled exception runtime error.

How would you write your main() and bar() functions?

```cpp
// main() function here
// complete the following
int main()
{
    try
    {
        ...
        bar();
        ...
    }
    catch(E7& e7)
    {
    }
    catch(...)
    {
        // do not rethrow!
    };
}

// bar() function here
// complete the following
void bar()
{
    ...
    ...
    try
    {
        foo();
    }
    catch(E3& e3)
    {
    }
    catch(E1& e1)
    {
    }
    catch(E2& e2)
    {
    }
    catch(...)
    {
        throw;
    };
    ...
    ...
}```
Q 8 [8 pts]:

(a) [2 pts] First recall that catch statements should receive exception objects by reference. One reason for this is efficiency. What is the other reason?

The other reason is to avoid object slicing in case the exception will be rethrown.

(b) [3 pts] Consider two exception classes BE and DE. The class DE inherits from BE through public inheritance. Consider three functions: Binky(), Pinky(), and Inky().

Binky() calls Pinky() and Pinky() calls Inky(). Inky() can throw either BE or DE.

What exceptions should Binky() and Pinky() be able to handle (or not handle) if you want to expose the bug that would result if you were catching exceptions by value and not by reference. Write the code that will expose the bug. Assume that exceptions will be handled by just cout-ing some text.

One possibility is that Binky() can only handle DE and Pinky() can only handle BE. Code corresponding to this scenario might appear as given below:

```cpp
// inside Binky()
try{
    ...
    Pinky();
    ...
}
catch(DE de)
{
    // exception rethrown from
    // catch(DE de) block in
    // Pinky() will not be
    // caught in this block
}
// inside Pinky()
```  

With above code, an exception of class DE raised by Inky() will go unhandled.

Another situation that may produce undesirable results is when Binky() is able to handle both DE and BE but Pinky() can not handle any of these exceptions and rethrows. In such case, if catch blocks in Pinky receive exceptions by value, Pinky() will always rethrow BE and Binky() will not be able to find out which exception, BE or DE, was originally thrown by Inky().

(c) [3 pts] Now consider a piece of code in a function Tinky() that can handle both DE and BE as given in above part. Tinky receives DE and BE in a single catch statement and within that catch block, prints out the correct type of exception. Write a piece of code that may accomplish this.

```cpp
void Tinky()
{
    // some code here
    try(Inky()) // may throw either BE or DE
        catch(BE& e) // single catch block
        {
            // You should be able to come here whether BE or DE is thrown.
            // In the following, print the correct type of exception.
            cout << "Exception received belonging to: ";
            cout << typeid(e).name() << endl;
        }
}
```
Q 9 [11 pts]:

(a) [2 pts] Provide the prototype for \texttt{operator=} which may be used with objects of class \texttt{T}. The prototype must be provided so that \texttt{operator=} is used as a member function. Be extremely careful with the input parameter and return type.

\begin{verbatim}
T& operator=(const T&);
\end{verbatim}

\texttt{const} with the return type is also acceptable (with reservation)!

(b) [1 pt] return is by reference or by value. Why?

Return by value would result in an extra undesirable call to the copy constructor. Hence, the return is by reference.

(c) [1 pt] return type is \texttt{const} or non-\texttt{const}? Why?

\texttt{non-const} so that a method can be called on the return type. e.g., \texttt{(a=b).foo()}. There is perhaps no reason to make the return type \texttt{const}.

(d) [1 pt] input type is \texttt{const} or non-\texttt{const}? Why?

\texttt{const} since we want to ensure that right hand object will not be modified.

(e) [1 pt] input type is by reference or by value? Why?

Input type is by reference for efficiency reasons (same reason as in part (b) above)

(f) [2 pts] The return value of expression \texttt{obj1 = obj2} is the same as \texttt{obj1} or \texttt{obj2} (which are set equal within the \texttt{operator=} class method. You can think of above expression being written as \texttt{obj1.operator=(obj2)}. That is, a method is called on \texttt{obj1} and \texttt{obj2} is passed as a parameter. Towards the end of your method implementation, you would like to return either \texttt{obj2} or \texttt{obj1} (which is available through \texttt{this} pointer). When you will be returning, hopefully \texttt{obj1} and \texttt{obj2} will be equal. Can we return either? Why? Why not?

If the return type is \texttt{const} (not preferable), then we can return either. However, if the return type is non-\texttt{const}, we are forced to return \texttt{obj1} because \texttt{obj2}, which was received as a \texttt{const} reference, can not be returned as a non-\texttt{const} reference.

(g) [2 pts] In the beginning of \texttt{operator=} implementation, we always put a check:

\begin{verbatim}
if(this != \&rho); // rho is the object on right hand of operator
\end{verbatim}

Why this check is placed?

This check is required to deal with self-assignment. If some heap memory is to be freed in the implementation of \texttt{operator=} , we must not do that in case of self-assignment (e.g., \texttt{myObject = myObject}).

(h) [2 pts] Assuming that the check in the above part is necessary, can we replace that check with:

\begin{verbatim}
if(*this != rho);
\end{verbatim}

Can we run into trouble? If yes, why?

We might run into trouble if we replace the check with the one stated here. The reason is that \texttt{operator!=} may not be defined for \texttt{rho} object. Notice that \texttt{operator!=} for pointers is always defined and so is \texttt{operator=} for all objects.
Q 10 [8 pts]: For the code written on the left, write the output on the right.

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

using namespace std;

class H
{
public:
  H() {cout << "H c'tor" << endl;}
};

class A
{
public:
  A() {cout << "A c'tor" << endl;}
};

class B:virtual public A
{
private:
  H h;
public:
  B() {cout << "B c'tor" << endl;}
};

class C:virtual public A
{
private:
  A a;
  B b;
public:
  C() {cout << "C c'tor" << endl;}
};

class D:public B, public C
{
public:
  D(): B() {cout << "D c'tor" << endl;}
private:
  string s1;
  H h;
  int i;
  B b;
  string s2;
};

int main()
{
  D d;
  return 0;
}
```
Q 11 [2 pts]: Wise guys always declare their destructors virtual. Why?

One should expect that programmers would inherit from their classes in future. One should also expect that people will use reference or pointer to their base class to refer or point to the derived class objects.

It is possible that additional encapsulated objects in the derived classes would be allocated on the heap (refer to class example of Thesis* in Grad class). The heap memory allocated in the derived class is most likely to be reclaimed in the derived class destructor which will be called through a base class reference or pointer only if the base class destructor is declared virtual. For example,

```cpp
{ ...  
    Student& s = new Grad();  
    ...  
}  // Desirable to call Grad() destructor for s here which is  
   // possible only if Student destructor is virtual
```

Q 12 [2 pts]: More wise guys avoid virtual functions because they are expensive. Why are virtual functions expensive?

Because calls to virtual functions need to made indirectly through a vptr in vtbl which adds more complexity to the function call and is, therefore, inefficient.

Q 13 [0 pts]: How many exercises we had in APT and what were their due dates?

Five, right?

When? Umm… well… who cares now!

Q 14 [1 pt]: Make a sketch of your instructor. Sketches of TAs not allowed!

HAVE A NICE WELL-DESERVED SPRING BREAK!