UEE1303(1048) S20
Final Examination

FULL SCORES:
120 %

EXAMINATION TIME:
18：30～21：30，total 180 minutes

INSTRUCTIONS:

This final examination is composed of 4 different problem sets. You are allowed to open any notes or books, but prohibited to browse on the internet to search C/C++ codes as direct answers. Read carefully the statements and requirements of each problem. Once you complete your program for one problem, please raise your hand and TA will come to you and test your program. Please note that points will be given until your program fully fulfills the requirements of each problem. No partial credits will be given.

You should be aware of that some lengthy problems are not as difficult as you think. In contrast, some short problems may not be solved easily. Last but not least, **cheating** is a serious academic demeanor and should be avoided at all most. Once any cheating is caught, all your answers will be void and get 0 point for your midterm.

Good luck!

UEE1303(1048) S20: Final Examination Demo Sheet

Student ID # : _____,Name :_____

Full Scores: up to 120 points
→ You may pick arbitrary numbers of problems to solve.

Problem Set 1 (60%)★		**Problem Set 2 (30%)★★**	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (15%)		Q1 (20%)	
Q2 (15%)			
Q3 (15%)		Q2(10%)	
Q4(15%)			

Problem Set 3 (30%)★★★		**Problem Set 4 (30%)★**	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (10%)		Q1 (30%)	
Q2 (10%)			
Q3 (10%)			

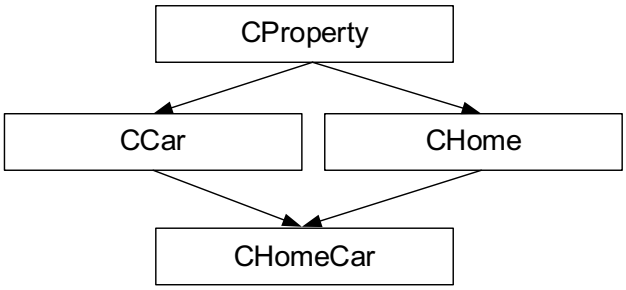
Total Score: _____ / 120

【SOURCE CODE AND TESTCASE】

The path of the source code and the open testcase for each problem set:
`<PATH> /tmp/OOP_Final/p1 ~ /tmp/OOP_Final/p4`
Copy the all the source codes and open testcases to your local directory and start to complete each problem in the corresponding folder.
`<EX> $ cp -R /tmp/OOP_Final ~/.`

【PROBLEM SET 1】(60%) ★ 60 LINES

Please refer to the inheritance figure shown as below:



Design a Property class (**CProperty**) that contains the data members for the owner (**owner**) in c-string (char*) and the price (**price**) in integer as an abstract base class (with a virtual function **computePrice()**). One class, **CCar**, inherits **CProperty** and contains two more properties for motor vehicles: the fuel capacity (**capacity**) and the travel distance (**distance**), both in double. Class **CHome** that also inherits **CProperty** includes two integer members for the number of floors (**floor**) and the land (**area**). The derived class **CHomeCar** inheriting both **CCar** and **CHome** maintains only one copy of **owner** and **price** with an extra member **seat** in integer and is used to declare the actual objects.

- A pure virtual function **computePrice()** declared in **CProperty** will be specified in **CCar**, **CHome** and **CHomeCar**, respectively.
  - ✓ In **CCar**, price = capacity*1000
  - ✓ In **CHome**, price = floor*100000*area
  - ✓ In **CHomeCar**, price=seat*50000+area*80000+capacity*2000

You are asked to complete the following functions for 4 tasks:

1. Define an abstract base class named **CProperty** with the data members, **owner** in string and **price** in integer where one pure virtual function **computePrice()** is declared publicly.

2. Define two classes named **CCar** and **CHome** both derived from **CProperty**. Properly redefine the above virtual functions **computePrice()** for computing the price for respective classes.
3. Define a class named **CHomeCar**, which publicly inherits both **CCar** and **CHome** but only maintains one copy of **owner** and **price**. Additionally, in such class, a data member **seat** in integer needs to be defined properly for the number of passengers. **CHomeCar** has its definition of **computePrice()**, different from those in **CCar and CHome**.
4. Realize the polymorphism by the dynamic binding on the pointer/reference variables among these related classes.

Example:

Q1 (15%)

The main program tries to declare a pointer ***x*** and a regular variable ***y*** of the abstract class **CProperty**. You need to show the compilation error, which indicates that y cannot be declared of abstract type **CProperty**. So you need to described **computePrice()** as a **pure virtual function** in **CProperty**.

```
#ifdef P1T1
   CProperty *x;
   CProperty y;
#endif
```

```
>make t1
g++ -ot1 -DP1T1 P1.cpp main.cpp
main.cpp: In function 'int main()':
main.cpp:10:11: error: cannot declare variable 'y' to be
of abstract type 'CProperty'
...
make: *** [Makefile:2: t1] Error 1
>
```

Q2 (15%)

The main program declares one regular variable ***p*** of **CCar** and the other ***q*** of **CHome**. Both classes are publicly derived from **CProperty**. So you need to define the pure virtual function **computePrice()** in **CProperty**. The definitions of **computePrice()** are given in the problem description. Note that many derived data members are inaccessible and you may write proper **getXXX()** functions in the base class. Constructors with parameters also need to be completed. Last, the **operator<<** functions also need to properly defined for both classes.

```
#ifdef P1T2
    CCar p("Joe", 2000, 100); //owner, capacity, distance
    CHome q("Tom", 3, 40); //owner, floor, area
    cout << p << q;
#endif
```

```
>make t2
g++ -ot2 -DP1T2 P1.cpp main.cpp
>./t2
Joe's car runs 100Km and costs $2M.
Tom's home has 3 floor(s) and costs $12M.
>
```

Q3 (15%)

The main program declares one regular variable ***r*** of **CHomeCar**, initialized by a constructor with parameters of ***owner***, ***capacity***, ***distance***, ***floor***, ***area*** and ***seat***. In **CHomeCar**, ***price***=***seat****50000+***area****80000+***capacity****2000. Proper **getXXX()** functions need to be defined in **CCar** and **CHome**. The **operator<<** functions also need to properly defined for **CHomeCar**. Note that since **CHomeCar** inherts **CCar** and **CHome**, both derived from **CProperty**, you need to make **CProperty** a virtual base class as well. So that only one copy of data members, **owner** and **price**, will exist in **CHomeCar**.

```
#ifdef P1T3
    CHomeCar r("Dan", 1000, 5000, 1, 8, 6);
    //initialized with owner, capacity, distance,
    // floor, area and seat
    cout << r;
#endif
```

```
>make t3
g++ -ot3 -DP1T3 P1.cpp main.cpp
>./t3
Dan's homecar has 6 seat(s), 1 floor(s), 1000 c.c.,
8 sqft, runs 5000 Km and costs $2.94M.
>
```

Q4 (15%)

In this task, the main program demonstrates the polymorphism by dynamic binding. Be aware of the **constant objects**! You need to make some of your member functions into **constant functions**, including the pure virtual function **computePrice()** in **CProperty**.

```
#ifdef P1T4
    const CHomeCar r("Bob", 2000, 15000, 2, 4, 4);
    const CProperty *z = &r;
    cout << z->getOwner() << "\'s property: $"
         << z->computePrice()/1000000.0 << "M\n";
    z = new CHome("Kim", 5, 20);
    cout << z->getOwner() << "\'s property: $"
         << z->computePrice()/1000000.0 << "M\n";
#endif
```

```
>make t4
g++ -ot4 -DP1T4 P1.cpp main.cpp
>./t4
Bob's property: $4.52M
Kim's property: $10M
>
```

==========

【PROBLEM SET 2】HASH TABLE (30%) ★★ 170 LINES

In computing, a hash table is a data structure that implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. During lookup, the key is hashed and the resulting hash indicates where the corresponding value is stored.
Ideally, the hash function will assign each key to a unique bucket, but most hash table designs employ an imperfect hash function, which might cause hash collisions where the hash function generates the same index for more than one key. Such collisions are always accommodated in some way. (You can see more details by surfing wiki.)

【The Program You Have to Complete】
In this problem set, we have three different type of hash table (integer, string, Point2D), you have to complete. The hash function is:

$$h(k) = k \% 10, k \in Z$$

There are three different definitions to get k each for integer, string, Point2D.
For integer, k = integer number. EX, int n1, k; k = n1;
For string, k = the sum of each character ASCII number. EX, string n1 = “abc”;

int k; k = ‘a’+‘b’+‘c’;
For Point2D, k = the sum of square from Point2D to origin. EX, Point2D n1(2, 3);
int k; k = 2^2+3^2;

Collision will happen when h(k1) = h(k2). It means the data will put in the same slot. Chaining is a method to solve collision. It uses link list to store the data in the same slot.

You should complete the `class Point2D` and template class `class Hash_table`.

// class Point2D

```
class Point2D
{
private:
    int x;
    int y;
public:
    //add constructors if needed

    //set x first, then set y
    void SetXY(int, int);

    //add any member function if needed
};
```

// class Hash_table

```
template <class T>
class Hash_table{
    class Hash{
        public:
            //add constructors if needed
            T value;
            Hash *next;

            //add any member function or variable if needed
    };

    private:
        //key to access hash table
        int key;
        //hash table
        Hash *head[10];
    public:
        //add constructors if needed
        //Part1 Q1 (20%)
        //insert to the tail.
        void insert(T);
```

```
        //search if data exist
        void search(T);
        //show all the table
        void show();

        //Part2  Q2 (10%)
        // insert and sort
        //void insertionsort(T);


        //add any member function or variable if needed
};
```

The `SetXY` function is first to set x variable, then set y variable.

The `value` is the data store in hash table.

The `key` applies hash function to get index of hast table.

The `head` is hash table.

The `insert` function insert data to hash table.

If collision happen, insert to the tail of linked list.

The `insertionsort` function inert data to hash table.

If collision happen, insert with specific rules of linked list.

For integer, it sorts from small to large.

For string, it sorts by dictionary order.

For Point2D, it sorts from smaller x to larger x. If equaling, sorted by y.

For each hash table, it will only apply one of `insert` or `insertionsort` function.

The `search` function finds if it exists the data in the hash table. The format is shown in result below.

The `show` function shows all the data in hash table, it shows from index 0 to 9, and linked list from head to tail. The format is shown in result below.

Execute `$` **`./pg01 pg01int.txt pg01string.txt pg01Point2D.txt`**

`//pg01int.txt`

```
1
11
41
21
31
```

```
32
```

//pg01string.txt

```
abm
awe
axy
axe
abc
abcabcabc
abcabc
abw
```

//pg01Point2D.txt

```
2 3
2 1
2 4
3 1
2 2
2 5
2 6
2 7
2 9
```

// pg01Part1.cpp //Q1

```
int main(int argc, char *argv[]) {
    Hash_table<int> inttable;
    Hash_table<string> strtable;
    Hash_table<Point2D> Point2Dtable;
    ifstream intfile(argv[1]);
    int int_data, x, y;
    string string_data;
    Point2D Point2D_data;

    while(!intfile.eof()) {
        intfile >> int_data;
        inttable.insert(int_data);
    }
    intfile.close();
    inttable.show();
    cout << endl;
    ifstream strfile(argv[2]);
    while(!strfile.eof()) {
        strfile >> string_data;
        strtable.insert(string_data);
    }
    strfile.close();
```

```
    strtable.search("abw");
    strtable.show();
    cout << endl;

    ifstream Point2Dfile(argv[3]);
    while(!Point2Dfile.eof()) {
        Point2Dfile >> x >> y;
        Point2D_data.SetXY(x, y);
        Point2Dtable.insert(Point2D_data);
    }
    Point2Dfile.close();
    Point2Dtable.search(Point2D(2,3));
    Point2Dtable.show();

    return 0;
}
```

Only display the index with value

// pg01Part1 result

```
index: 1
1 11 41 21 31
key: 2
32

abw exists.
index: 2
abcabcabc
index: 4
abm abc abw
index: 7
awe
index: 8
axy axe abcabc

(4, 1) does not exist.
index: 0
(2, 4) (3, 1) (2, 6)
index: 3
(2, 3) (2, 7)
index: 5
(2, 1) (2, 9)
index: 8
(2, 2)
index: 9
(2, 5)
```

// pg01Part2.cpp //Q2

```
int main(int argc, char *argv[]) {
    Hash_table<int> inttable;
    Hash_table<string> strtable;
    Hash_table<Point2D> Point2Dtable;
    ifstream intfile(argv[1]);
    int int_data, x, y;
    string string_data;
    Point2D Point2D_data;

    while(!intfile.eof()) {
        intfile >> int_data;
        inttable.insertionsort(int_data);
    }
    intfile.close();
    inttable.show();
    cout << endl;
    ifstream strfile(argv[2]);
    while(!strfile.eof()) {
        strfile >> string_data;
        strtable.insertionsort(string_data);
    }
    strfile.close();
    strtable.search("abw");
    strtable.show();
    cout << endl;

    ifstream Point2Dfile(argv[3]);
    while(!Point2Dfile.eof()) {
        Point2Dfile >> x >> y;
        Point2D_data.SetXY(x, y);
        Point2Dtable.insert(Point2D_data);
    }
    Point2Dfile.close();
    Point2Dtable.search(Point2D(2,3));
    Point2Dtable.show();

    return 0;
}
```

// pg01Part2 result

```
index: 1
1 11 21 31 41
index: 2
32

abw exists.
index: 2
abcabcabc
index: 4
abc abm abw
```

```
index: 7
awe
index: 8
abcabc axe axy

(4, 1) does not exist.
index: 0
(2, 4) (2, 6) (3, 1)
index: 3
(2, 3) (2, 7)
index: 5
(2, 1) (2, 9)
index: 8
(2, 2)
index: 9
(2, 5)
```

==========

【PROBLEM SET 3】FILE SYSTEM SIMULATOR (30%) ★★★220 LINES

In this simulator, you should implement some linux-like command to do file operations.

Q1 (10%)

echo:

echo hhh > a.txt //open a file named a.txt, put hhh into this file

echo aabbcc >> a.txt //open a file named a.txt, append aabbcc into the end of this file

cat:

cat a.txt //cat content of a.txt, you should print "txt file: ${content}“.

cat a.pdf //cat content of a.pdf, you should print "pdf format: ${content}"

ls:

ls //list information of all file and directory in current working directory, include permissions, size, owner, file name

ls a.txt//list information of a.txt

ls test_dir//list all file information in test_dir

mkdir:

mkdir test_dir //create directory

Q2 (10%)

chmod:

chmod 744 a.txt //change file or directory permissions

//7 = 4+2+1, r = 4, w = 2, x = 1

//744 = rwxr--r--

create:

create aabbcc a.pdf // you can use this command to create a file named a.pdf, content is aabbcc

cp:

cp a.txt b.txt //copy content, information of a.txt into b.txt

Q3 (10%)

chown:

chown user0 a.txt //change file or directory ownership

cd:

cd test_dir //go into test_dir directory if it exists in current working directory

cd .. //go to the parent of current working directory

***

Notice:

create command can create a new pdf or txt file, but you cannot use echo command to write pdf file.

The permission of directory or file is "rwxrwxrwx" by default.

The owner of directory or file is "root" by default.

The size of a character is 1, ex: "abcde"->size is 5 by default.

If command operations are not allowed, you should print the error message.

You should implement all error handler in the test case, we will not have another error handler you have never seen in the hidden test case.

You should use overwrite method to do read, write function in different class.

You cannot add or modify variable in class, but you can add or modify function.

You should print each command line and print $ before.

***

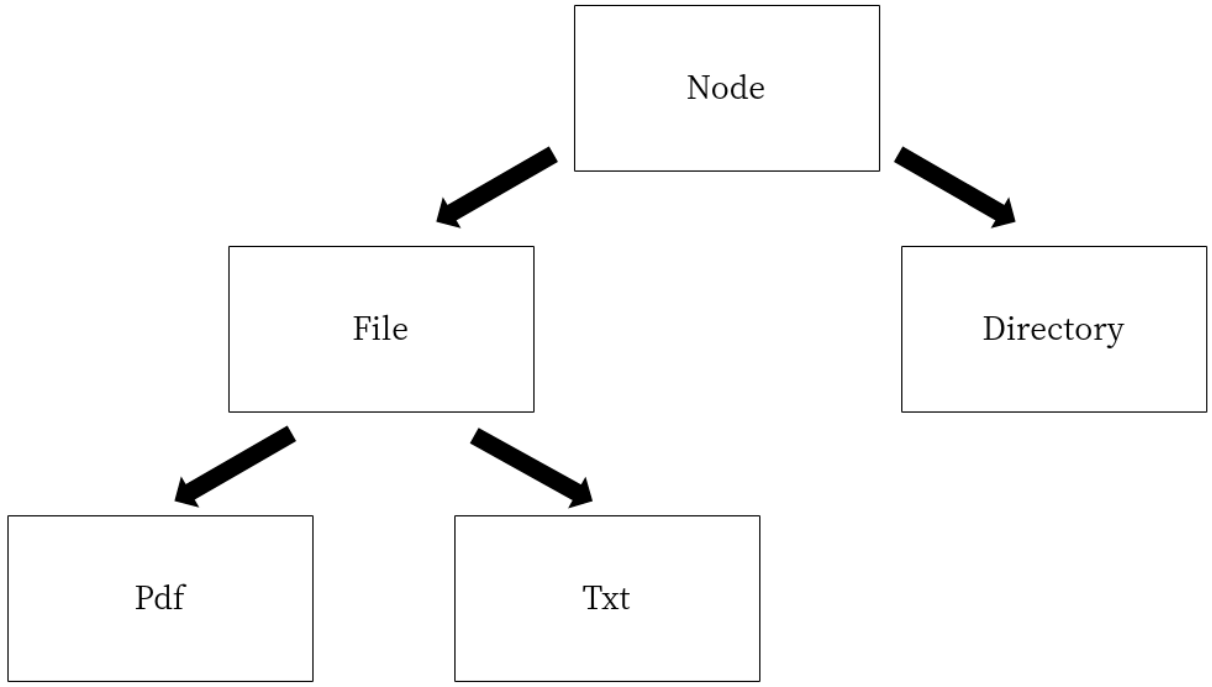
Compile:

g++ main.cpp Node.cpp File.cpp Pdf.cpp Txt.cpp Directory.cpp -o file_sys.o

Execution:

./file_sys.o ./inst${test_number}.in

Structure:



// pg03Part1 input(inst1.in)

```
echo 999 > 1.txt
cat 1.txt
echo 544 > 2.txt
cat 2.txt
ls
echo 8866 >> 1.txt
cat 1.txt
echo 9999 > 2.txt
ls
mkdir test_dir
echo hhh > test_dir
ls 1.txt
cat 8.txt
ls 8.txt
```

// pg03Part1 result

```
$echo 999 > 1.txt
$cat 1.txt
txt file: 999
$echo 544 > 2.txt
$cat 2.txt
```

```
txt file: 544
$ls
total file number is: 2
rwxrwxrwx 3 root 1.txt
rwxrwxrwx 3 root 2.txt
$echo 8866 >> 1.txt
$cat 1.txt
txt file: 9998866
$echo 9999 > 2.txt
$ls
total file number is: 2
rwxrwxrwx 7 root 1.txt
rwxrwxrwx 4 root 2.txt
$mkdir test_dir
$echo hhh > test_dir
error: you should not write directory
$ls 1.txt
rwxrwxrwx 7 root 1.txt
$cat 8.txt
error: cannot find this file.
$ls 8.txt
error: cannot find this file.
```

***** Part2 and Part3 you can see in public testcase. *****

==========

【PROBLEM SET 4】STL (30%) ★ 100 LINES

Please complete `func.h` and `func.cpp` to make `main.cpp` work properly.
The program reads in a sequence of integers and uses vector of stacks to store the data. The storage of data should follow the rules below.

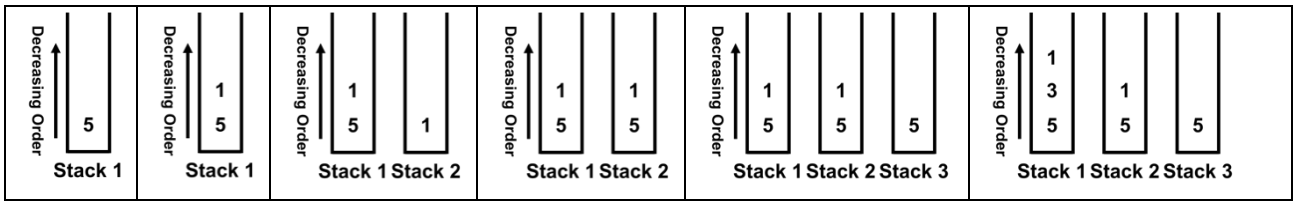
1. Each integer in the sequence would only be pushed into one stack in the vector.
2. The elements in each stack should not be repeating. If there is the same element already in the stack, the read-in element should be pushed into the next stack or a new-created stack (If all the existing stacks have already contained the read-in element, please create a new stack to push the read-in element and push the new-created stack at the end of the vector).
3. For each stack in the vector, the elements in the stack should be placed in decreasing order. If the read-in element is smaller than the top element of the stack, please pop the top element out until the read-in element could be pushed into. And push the popped elements back into the stack.
4. Please display elements in each stack in the vector from top to bottom as example below shows.

Example:

Command: $ ./p4 <testcase>

//t1.in

```
5 1 1 5 5 3
```



//Result

```
Stack 1: 1 3 5
Stack 2: 1 5
Stack 3: 5
```

Source Code:

//main.cpp **DO NOT MODIFY**

```
#include<iostream>
#include<string>
#include”func.h”

int main(int argc, char** argv)
{
    vector<stack<int> > v;
    parse(argv[1],v);
    display(v);
    return 0;
}
```

//func.h **YOU CAN ONLY ADD FUNCTION DECLARATION**

```
#ifndef _FUNC_H_
#define _FUNC_H_
#include<stack>
#include<vector>
using namespace std;
void parse(char*, vector<stack<int> >&);
void display(vector<stack<int> >);
#endif
```