【PROBLEM SET A】 (60%)

In mathematics, a matrix (plural matrices) is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns, which is used to represent a mathematical object or a property of such an object. Matrix calculation is widely used in computer science and electrical engineering domains, such as graphics rendering, deep learning, and circuit simulation.

In this problem set, you need to implement a Matrix class with basic matrix calculations and algorithms.

- ✔ The declaration of class **Matrix** is in **Matrix.h**.
- ✔ The main programs are in **A-1.cpp, A-2.cpp, A-3.cpp**. **DO NOT MODIFY THE MAIN PROGRAMS.**
- ✔ You can modify and add proper prototypes (hint: beware of **const**) to **Matrix.h**.
- ✔ To pass the test, your program cannot contain memory leaks. You can use the following command to test for memory leaks.

  > valgrind <your_execution_binary>
  ex: valgrind ./A-1

  Valgrind reference:
  https://web.stanford.edu/class/archive/cs/cs107/cs107.1234/resources/valgrind.html
- ✔ You can copy **A-1.cpp**, **A-2.cpp**, **A-3.cpp**, and **Matrix.h** from /home/share/midterm/A/.
- ✔ Compile

  Level 1 (20%):
  g++ A-1.cpp Matrix.cpp -o ./A-1
  Level 2 (20%):
  g++ A-2.cpp Matrix.cpp -o ./A-2
  Level 3 (20%):
  g++ A-3.cpp Matrix.cpp -o ./A-3

```
//Matrix.h
class Matrix {
 private:
   double** data;
   int row;
   int col;
 public:
    // Level 1
    // Constructors and destructor
    Matrix(); // Set row and col to 0, data to nullptr
    Matrix(int row, int col); // Create a zero matrix
    Matrix(int row, int col, double** arr2D); // Copy from array
    Matrix(const Matrix& matrix); // Copy from another matrix
```

```
~Matrix(); // Delete data

    // Setters and Getters
    // Set the value of the given row and col
    void setValue(int rowIdx, int colIdx, double value);
    // Get the value of the given row and col
    double getValue(int rowIdx, int colIdx) const;
    // Transpose
    Matrix transpose() const; // Return a new transposed matrix

    // Operator Overload
    // Overload =, +=, -=, +, -, +m, -m, ==, !=, ostream<<
    // operator=();   // Copy to a new matrix
    // operator+=();  // Matrix addition, ex: a += b;
    // operator-=();  // Matrix subtraction, ex: a -= b;
    // operator+();   // Matrix addition, ex: c = a + b;
    // operator-();   // Matrix subtraction, ex: c = a - b;
    // operator+();   // Positive Matrix, ex: Matrix b = +a;
    // operator-();   // Negative Matrix, ex: Matrix b = -a;
    // operator==();  // Return true if the two matrix is equal
    // operator!=();  // Return false if the two matrix is equal
    // operator<<();  // Output the matrix, ex: cout << a;

    // Level 2
    // Overload *=, *
    // operator*=(Matrix); // Matrix and Matrix multiplication
    // operator*=(double); // Matrix and double multiplication
    // operator*(Matrix); // Matrix and Matrix multiplication
    // operator*(double); // Matrix and double multiplication
    // operator*();       // double and Matrix multiplication

    // Level 3
    double determinant() const; // The determinant of the matrix

};
```

A-01 LEVEL 1 (20 points)

Please complete all constructors, destructor, the required operators, and the required member functions in each above class declaration with the following requirements. You can reference **`level1.cpp`** to learn about the content of the main function. You don't have to check the dimension of the matrix.

Reference:
https://en.wikipedia.org/wiki/Matrix_(mathematics)#Basic_operations.

(1) There are four constructors to create **`Matrix`** objects: **`Matrix(), Matrix(int, int), Matrix(int, int, double**), `** and **`Matrix(const Matrix&)`**. For copy constructor, you have to **hard copy** the original matrix.

(2) In `~Matrix()`, you have to delete the dynamic array you allocated. We will check your program for memory leaks.

(3) `setValue(int, int, double)`, a mutator function, use to set values to private data members. Set the value of the given row and column.

(4) `getValue(int, int)`, an accessor function, use to read from private data members. Get the value of the given row and column.

(5) `transpose()`, return the transpose of the original matrix.

(6) `operator=`, **hard copy** the original matrix to a new matrix.

(7) `operator+, operator-, operator+=, operator-=`, calculate the addition or subtraction of two matrices. Remember to use the proper function prototypes (hint: beware of const)

(8) `operator==` and `operator!=`, need to check col, row, and values in data

(9) `operator<<` is a given friend member function that overloads the operator `<<` to output the private data member to ostream `out`.

The sample output of level 1:

```
>./A-1
a =
1 0
0 2

b =
0.5 3.2
1.6 -0.3

a + b =
1.5 3.2
1.6 1.7

a - b =
0.5 -3.2
-1.6 2.3

+b =
0.5 3.2
1.6 -0.3

-b =
-0.5 -3.2
-1.6 0.3

a.transpose() =
1 0
0 2
```

```
b.transpose() =
0.5 1.6
3.2 -0.3

c = a
d = -b
c += a + b
d -= c + a
c =
2.5 3.2
1.6 3.7

d =
-4 -6.4
-3.2 -5.4

d = c
(a != b) = 1
(c != d) = 0
(c == d) = 1
e =
0 0 0
0 0 3

f =
0 0
0 0
0 -2

e.transpose() =
0 0
0 0
0 3

f.transpose() =
0 0 0
0 0 -2

f = e
f =
0 0 0
0 0 3

e = a
e =
1 0
0 2

>
```

A-2 LEVEL 2 (20 points)

Please complete the following overloading operators * and *= to calculate matrix multiplication and matrix scaler multiplication. You don't have to check the dimension of the matrix.

Reference: https://en.wikipedia.org/wiki/Matrix_multiplication.

(1) For operator *, you have to overload three types of multiplication, **Matrix * Matrix**, **Matrix * double**, and **double * Matrix**.

(2) For operator *=, you have to overload two types of multiplication, **Matrix *= Matrix** and **Matrix *= double**.

The sample output of level 2:

```
>./A-02
a =
1 0
0 2

b =
0.5 3.2
1.6 -0.3

c = 1.7

a * b =
0.5 3.2
3.2 -0.6

b * a =
0.5 6.4
1.6 -0.6

a * c =
1.7 0
0 3.4

c * b =
0.85 5.44
2.72 -0.51

a *= b * a * c

a =
0.85 10.88
5.44 -2.04

a *= -c
a =
```

```
-1.445 -18.496
-9.248 3.468

d =
1 2 3
4 5 6

e =
1 2
3 4
5 6

d * e =
22 28
49 64

e * d =
9 12 15
19 26 33
29 40 51

>
```

A-3 LEVEL 3 (20 points)

Please complete the **`determinant()`** function to calculate the determinant of a square matrix. You don't have to check whether the matrix is square.

Reference: https://en.wikipedia.org/wiki/Determinant
https://www.youtube.com/watch?v=Ip3X9LOh2dk

The sample output of level 3:

```
>./A-03
a =
1.2 2.2 3.2 3.2
4.2 5.2 6.2 6.2
-2.3 5.2 6.3 6.2
-3.8 5.9 6.2 8.3

a.determinant() = 43.14
>
```

【PROBLEM SET B】 (60%)

A Container is any receptacle or enclosure for holding a product used in storage, packaging, and transportation, including shipping.

In computer science, an associative array, map, symbol table, or dictionary is an abstract data type that stores a collection of (key, value) pairs, such that each possible key appears at most once in the collection. In mathematical terms, an associative array is a function with a finite domain. It supports 'lookup', 'remove', and 'insert' operations.

Therefore, a container is a class or a data structure whose instances are collections of other objects in computer science. In other words, they store objects in an organized way that follows specific access rules.

In this problem set, you need to implement a Container by yourself with two classes, including Pair and Container.

- ✔ The declaration of class **`Pair`** and class **`Container`** is separately in **`Pair.h`** and **`Container.h`**.
- ✔ You only need to separately define two classes in **`Pair.cpp`** and **`Container.cpp`**.
- ✔ The main programs are in **`B-1.cpp`**, **`B-2.cpp`**. **DO NOT MODIFY THE MAIN PROGRAMS**.
- ✔ Do not modify and add any function and prototype to **`Pair.h`** and **`Container.h`**.
- ✔ To pass the test, your program cannot contain memory leaks. You can use the following command to test for memory leaks.

  ```
  > valgrind <your_executable_file> <arguments_if_needed>
  ex: valgrind ./B-1 1.dat 3
  ```

  Valgrind reference:
  https://web.stanford.edu/class/archive/cs/cs107/cs107.1234/resources/valgrind.html
- ✔ You can copy two main files and header files from /home/share/midtem/B/.
- ✔ Compile

  Level 1 (40%):

  ```
  g++ B-1.cpp Pair.cpp Container.cpp -o ./B-1
  ```

  Level 2 (20%):

  ```
  g++ B-2.cpp Pair.cpp Container.cpp -o ./B-2
  ```

B-1 LEVEL 1 (40 points)

As we all know, each container has its own set of rules that must be followed. Now, let's define our **Pair** class and a portion of the **Container** class to build our Container in level 1.

Please complete the **Pair** class of all constructors, destructor, accessors, mutators, static functions, and operator overloads in below **Pair** class declaration with the following requirements.

- ✓ There are four constructors to create **Pair** objects: **Pair(), Pair(int), Pair(int,int),** and **Pair(const Pair &)**. For copy constructor, you have to **hard copy** the original pair.
- ✓ In constructors and destructor, you have to maintain the static int **count** in **Pair** class to ensure how many classes exist.
- ✓ **Operator*=(Pair &)**, a private operator, use to support the sort() function in **Container** class for swapping two Pairs. Because it is the Pair's own functionalities for Container, the **operator*=** needs to declare in the private area instead of the public area.
- ✓ **setPair(int, int)**, a mutator function, use to set values to private data members. Set the value of the given key and value separately in private data members of **first** and **second**.
- ✓ **getKey()**, an accessor function, use to read from private data members. Return the key of this pair.
- ✓ **getValue()**, an accessor function, use to read from private data members. Return the value of this pair.
- ✓ **getCount()**, a static function, use to access private static data. Return the **Pair** class **count**.
- ✓ **readBinaryFile(char *, int)**, static function, use to read binary file in main function by returning dynamic Pair array. Need to construct a new Pair as you read a pair data from the binary file.
- ✓ **operator=**, **hard copy** the original pair to a new pair.
- ✓ **operator<**, return Boolean for friend class **Container** to do sorting.
- ✓ **operator<<,** output the private data member of **first** and **second** to ostream.

```
#ifndef _PAIR_H_
#define _PAIR_H_
#include <iostream>

class Pair
{
private:
    int first;
    int second;
    static int count;
    void operator*=(Pair &);

public:
    friend class Container;
    // Level 1 25%
    // Constructors and destructor
    Pair();
    Pair(int);
    Pair(int, int);
    Pair(const Pair &);
    ~Pair();

    // Accessor and mutators
    void setPair(int, int);
    int getKey();
    int getValue();

    // Static functions
    static int getCount();
    static Pair **readBinaryFile(char *, int);

    // Operator overloads
    Pair &operator=(const Pair &);
    bool operator<(const Pair &);
    friend std::ostream &operator<<(std::ostream &, const Pair);
};
#endif
```

Please complete the **Container** class of all constructors, destructor, member functions, static functions, and friend functions in below **Container** class declaration with the following requirements.

- ✓ There are four constructors to create **Container** objects: **Container(), Container(const Pair &), Container(Pair **, int),** and **Container(const Container &)**. For copy constructor, you must **hard copy** the original container.
- ✓ In constructors and destructor, you have to maintain the static int count in **Container** class to ensure how many classes exist and dynamic array you allocated. We will check program for **memory leaks**.
- ✓ **Sort()**, a private member function, use to sort the Pairs in the container **by default**. Because it is the container's own characteristics, the sort function needs to declare in the private area instead of the public area.
- ✓ **Erase(Pair *)**, a member function, use to erase the Pair to which the argument points.
- ✓ **Clear(),** a member function, use to remove all pairs in container.
- ✓ **Empty(),** a member function, use to return whether the container is empty or not.
- ✓ **Insert(const Pair &),** a member function, use to add the given pair to the container. Keep in mind that if you insert the same key, it will be disregarded.
- ✓ **Size(),** a member function, use to return the container's **capacity**.
- ✓ **Find(int),** a member function, use to find whether the given value of key is stored in the container or not. If the pair exists, return its address; otherwise, return NULL.
- ✓ **End(),** a member function, usually return the next Pair address of the last pair in the container. In this problem, you only need to return NULL.
- ✓ **Begin(),** a member function, use to return the address of first Pair in the container if the value of capacity is larger than zero; otherwise, return NULL.
- ✓ **getCount(),** a static function, use to access private static data. Return the Container class **count**.
- ✓ **Operator<<,** output the private data member of all pairs to ostream.

```
//Container.h
#ifndef _CONTAINER_H_
#define _CONTAINER_H_
#include "Pair.h"

class Container
{
private:
    Pair **data;
    int capacity = 0;
    static int count;
    void sort();

public:
    // Level 1 25%
    // Constructors and destructor
    Container();
    Container(const Pair &);
    Container(Pair **, int);
    Container(const Container &);
    ~Container();

    // Member functions
    void erase(Pair *);
    void clear();
    bool empty();
    bool insert(const Pair &);
    int size() const;
    Pair *find(int);
    Pair *end();
    Pair *begin();

    // Static & friend functions
    static int getCount();
    friend std::ostream &operator<<(std::ostream &, const
Container &);

    // Level 2 15%
    // Union container
    Container &operator+=(const Container &);
    Container operator+(const Container &);
    // Complement container
    Container &operator-=(const Container &);
    Container operator-(const Container &);
    // Swap container
    void operator*=(Container &);
    // Assign container
    Container &operator=(const Container &);
};
```

```
#endif
```

You can reference **B-1.cpp** to learn about the content of the main function.

```
>cat B-1.cpp
#include <iostream>
#include <fstream>
#include "Pair.h"
#include "Container.h"
using namespace std;

int main(int argc, char *argv[])
{
     if (argc < 3)
     {
          cout << "Usage: ./B-1 <input_file> <number_of_pairs>"
<< endl;
          return -1;
     }
     Pair **Pptr;
     Pptr = Pair::readBinaryFile(argv[1], atoi(argv[2]));
     cout << "Pptr:" << endl;
     for (int i = 0; i < atoi(argv[2]); i++)
          cout << *Pptr[i];
cout << "The total number of Pairs: " << Pair::getCount() <<
endl
          << endl;

     Container C1(Pptr, atoi(argv[2]));
     cout << "The total number of Pairs before delete: " <<
Pair::getCount() << endl
          << endl;

     for (int i = 0; i < atoi(argv[2]); i++)
          delete Pptr[i];
     delete[] Pptr;
     cout << "Container C1:" << endl
          << C1;
     cout << "The total number of Pairs after delete: " <<
Pair::getCount() << endl
          << endl;

     Container C1_copy(C1);
     cout << "Container C1_copy:" << endl
          << C1_copy
          << endl;
     cout << "The total number of Containers: " <<
Container::getCount() << endl
```

```
Container::getCount() << endl
          << endl;

     // repeat insert
     C1_copy.insert(Pair(4, 900));
     cout << "C1_copy after repeat insert:" << endl
          << C1_copy
          << endl;

     Pair p1(3);
     Container C2(p1);

     Pair p2(2, 5);
     C2.insert(p2);

     C2.insert(Pair(1, 9));
     cout << "Container C2:" << endl
          << C2;
     cout << "The total number of Pairs: " << Pair::getCount()
<< endl
          << endl;
     cout << "The total number of Containers: " <<
Container::getCount() << endl
          << endl;

     Pair *ptr_1 = C2.find(1);

     if (ptr_1 != C2.end())
          cout << "value of ptr_1: " << ptr_1->getValue() <<
endl
               << endl;
     else
          cout << "ptr_1 not found" << endl
               << endl;

     Pair *ptr_4 = C2.find(4);

     if (ptr_4 != C2.end())
          cout << "value of ptr_4: " << ptr_4->getValue() <<
endl
               << endl;
     else
          cout << "ptr_4 not found" << endl
               << endl;

     cout << "C2 before erase:" << endl
          << C2
          << endl;
     Pair *run;
```

```
        while (!C2.empty())
        {
                run = C2.begin();
                cout << "ready to erase " << *run;
                C2.erase(run);
        }
        cout << "Container C2:" << endl
                << C2
                << endl;

        if (!C2.empty())
                C2.clear();
        else
                cout << "C2 is empty!" << endl;

        cout << "C1 before clear:" << endl
                << C1;
        C1.clear();
        cout << "C1 after clear:" << endl
                << C1;

        // two pair object + one C1_copy container with three pair
object
        cout << "The total number of Pairs before end: " <<
Pair::getCount() << endl
                << endl;
        // three container object
        cout << "The total number of Containers before end: " <<
Container::getCount() << endl
                << endl;

        if (C1.empty())
                cout << "Success" << endl;
        else
                cout << "Fail in clear()" << endl;

        return 0;
}
>
```

The sample output of level 1:

```
>./B-1 1.dat 3
Pptr:
key: 6, value: 10
key: 5, value: 50
key: 4, value: 700
The total number of Pairs: 3

The total number of Pairs before delete: 6

Container C1:
key: 4, value: 700
key: 5, value: 50
key: 6, value: 10
The total number of Pairs after delete: 3

Container C1_copy:
key: 4, value: 700
key: 5, value: 50
key: 6, value: 10

The total number of Containers: 2

C1_copy after repeat insert:
key: 4, value: 700
key: 5, value: 50
key: 6, value: 10

Container C2:
key: 1, value: 9
key: 2, value: 5
key: 3, value: 0
The total number of Pairs: 11

The total number of Containers: 3

value of ptr_1: 9

ptr_4 not found

C2 before erase:
key: 1, value: 9
key: 2, value: 5
key: 3, value: 0

ready to erase key: 1, value: 9
ready to erase key: 2, value: 5
ready to erase key: 3, value: 0
Container C2:
```

```
C2 is empty!
C1 before clear:
key: 4, value: 700
key: 5, value: 50
key: 6, value: 10
C1 after clear:
The total number of Pairs before end: 5

The total number of Containers before end: 3

Success
>
```

B-2 LEVEL 2 (20 points)

Please complete the following overloading operators in **Container** class to implement Container union and complement in level 2.

- ✓ **Operator+(const Container &)** and **operator+=(const Container &)**, union two containers. Note that if two containers have the same key, please add the value of latter container to the value of former container.
- ✓ **Operator-(const Container &)** and **operator-=(const Container &)**, complement two container. Subtract the value of the latter container from the value of the former container if two containers have the same key.
- ✓ **Operator*=(const Container &)**, swap two container.
- ✓ **Operator=(const Container &)**, **hard copy** the original container to a new container.

You can reference **B-2.cpp** to learn about the content of the main function.

```
>cat B-2.cpp
#include <iostream>
#include <fstream>
#include "Container.h"
#include "Pair.h"
using namespace std;

int main(int argc, char *argv[])
{
     if (argc < 3)
     {
          cout << "Usage: ./B-2 <input_file> <number_of_pairs>"
<< endl;
          return -1;
     }
```

```
    Pair **Pptr = Pair::readBinaryFile(argv[1], atoi(argv[2]));
    Container C1(Pptr, atoi(argv[2]));
    for (int i = 0; i < atoi(argv[2]); i++)
        delete Pptr[i];
    delete[] Pptr;
    cout << "Container C1:" << endl
        << C1;
    cout << "The total number of Pairs: " << Pair::getCount()
<< endl
        << endl;

    Container C2(Pair(4, -50));
    C2.insert(Pair(3, 199));
    C2.insert(Pair(1, 33));
    C2.insert(Pair(2, -70));
    cout << "Container C2:" << endl
        << C2;
    cout << "The total number of Pairs: " << Pair::getCount()
<< endl
        << endl;

    Container C3(C1 + C2);
    cout << "Container C3:" << endl
        << C3;
    cout << "The total number of Pairs: " << Pair::getCount()
<< endl
        << endl;

    Container C4(C1 - C2);
    cout << "Container C4:" << endl
        << C4;
    cout << "The total number of Pairs: " << Pair::getCount()
<< endl
        << endl;

    C3 *= C4;
    Container C5;
    C5 = C4;
    cout << "Container C5:" << endl
        << C5;
    cout << "The total number of Pairs: " << Pair::getCount()
<< endl
        << endl;
    cout << "The total number of Containers: " <<
Container::getCount() << endl;

    return 0;
}
>
```

The sample output of level 2:

```
>./B-2 2.dat 6
Container C1:
key: 1, value: -200
key: 2, value: -70
key: 4, value: 700
key: 5, value: 50
key: 6, value: 10
The total number of Pairs: 5

Container C2:
key: 1, value: 33
key: 2, value: -70
key: 3, value: 199
key: 4, value: -50
The total number of Pairs: 9

Container C3:
key: 1, value: -167
key: 2, value: -140
key: 3, value: 199
key: 4, value: 650
key: 5, value: 50
key: 6, value: 10
The total number of Pairs: 15

Container C4:
key: 1, value: -233
key: 2, value: 0
key: 4, value: 750
key: 5, value: 50
key: 6, value: 10
The total number of Pairs: 20

Container C5:
key: 1, value: -167
key: 2, value: -140
key: 3, value: 199
key: 4, value: 650
key: 5, value: 50
key: 6, value: 10
The total number of Pairs: 26

The total number of Containers: 5
>
```

【PROBLEM SET C】 (60%)

In computing, a **hash table**, also known as hash map, is a data structure that implements an associative array or dictionary. It is an abstract data type that maps 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 typically accommodated in some way.

In a well-dimensioned hash table, the average time complexity for each lookup is independent of the number of elements stored in the table. Many hash table designs also allow arbitrary insertions and deletions of key–value pairs, at amortized constant average cost per operation.

Hashing is an example of a space-time tradeoff. If memory is infinite, the entire key can be used directly as an index to locate its value with a single memory access. On the other hand, if infinite time is available, values can be stored without regard for their keys, and a binary search or linear search can be used to retrieve the element.

In many situations, hash tables turn out to be on average more efficient than search trees or any other table lookup structure. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches, and sets.

In this problem, please build a hash table and store item information in the hash table with the key of item number (`itemNum`).

- ✔ The item information should be stored in `struct "node"`, which is already defined in the given header files.

```
struct node
{
    int itemNum;
    int price;
    node* ptr;
};
```

- ✔ The input item information should be read as `struct "node"` with binary file, while every `node* ptr` of input data is pointed to `NULL` intially.
  `ex: $ ./C-1 1.dat`
- ✔ Please handle collision problem of hash table with linked list method.

C-1 LEVEL 1 (35 points)

Please implement the hash table with the hashing function of modulo-division method, and complete the **HashTable** class of all constructors, destructor, function members, static functions, and operator overloads in below **HashTable** class declaration with the following requirements.

- ✓ The declaration of class **HashTable** is in **HashTable-1.h**.
- ✓ You only need to define the **HashTable-1.cpp.**
- ✓ The main program is in **C-1.cpp**. **DO NOT MODIFY THE MAIN PROGRAMS.**
- ✓ Do not modify and add any function and prototype to **HashTable-1.h**
- ✓ You can copy the main file, header file and testcases from /home/share/midtem/C/L1/.
- ✓ Compile

```
g++ C-1.cpp HashTable-1.cpp -o C-1
```

- ✓ **tableSize** is used to store the size of the hash table with the unit of **node**.
- ✓ **tablePtr** should be point to the dynamic memory allocated for hash table.
- ✓ **HashingFunction** should be point to the desired hashing function for hash table.
- ✓ In constructor, You have to allocate dynamic memory for hash table
- ✓ In destructor, you have to release the memory of hash table.
- ✓ **operator += (node n)** is used to insert node data to hash table.
- ✓ **printTable()** is used to print the whole hash table on the terminal. The "Prime Area" column shows the first inserted node in that address, while the "Overflow List" column shows the nodes inserted after the first node and linked one by one in input order. The title of "Home Addr", "Prime Area" and "Overflow List" should be split by a tab character, while the below value of "Home Addr", "Prime Area" and "Overflow List" should be split by two tab characters. And the item number and the price should be split by '/'.
- ✓ **searchTable()**, ask the user to enter the item number, search for it in hash table, then print the price on the terminal.
- ✓ **analyzeTable()**, analyze and print out the information of hash table. The first information is the usage in the Prime Area, the second is the average nodes number in Overflow List, and the last is the longest length of the linked list in hash table.

- ✓ **moduloDivision(int key, int tableSize)**, hashing function for hash table, generate the address for node data with key value and table size by modulo-division method.

```
#ifndef HASHTABLE_H
#define HASHTABLE_H
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
using namespace std;
struct node
{
    int itemNum;
    int price;
    node* ptr;
};
class HashTable{
private:
    int tableSize;
    node* tablePtr;
    int (*HashingFuncton)(int key, int tableSize);
public:
    HashTable(int tableSize);
    ~HashTable();

    void operator += (node n); //Insert data to hash table
    void printTable();
    void searchTable();
    void analyzeTable();

    static int moduloDivision(int key, int tableSize); //Hashing
function
};
#endif
```

You can reference **C-1.cpp** to learn about the content of the main function.

The content of **C-1.cpp**:

```
>cat C-1.cpp
#include "HashTable-1.h"
using namespace std;

int main(int argc, char** argv){
    int tableSize, option;
    ifstream in(argv[1]);
    cout << "========== Level 1: Hash Table with Modulo-
division Hashing Function ==========" << endl;
    cout << "Please enter the size of hash table: ";
    cin >> tableSize;

    //Create hash table
    HashTable HT(tableSize);

    //Insert data to hash table
    node n;
    while(in.read((char*)&n,sizeof(n)))
        HT += n;

    //Print menu
    char sel;
    cout << "\ns) Search\np) Print\na) Analyze\nq) Quit\nYour
selection: ";
    cin >> sel;
    while(sel != 'q'){
        switch(sel){
            case 's':
                HT.searchTable();
                break;
            case 'p':
                HT.printTable();
                break;
            case 'a':
                HT.analyzeTable();
                break;
            default:
                break;
        }
        cout << "\ns) Search\np) Print\na) Analyze\nq)
Quit\nYour selection: ";
        cin >> sel;
    }
    return 0;
}
>
```

The sample output of level 1:

```
>./C-1 1.dat
==== Level 1: Hash Table with Modulo-division Hashing Function
====
Please enter the size of hash table: 10

s) Search
p) Print
a) Analyze
q) Quit
Your selection: p

Home Addr       Prime Area      Overflow List
0               130/30          150/50
1
2               112/12
3               173/17          133/13, 123/12, 143/14, 193/19
4
5
6               156/56          166/66, 176/76
7               197/19          167/16, 117/11
8
9               189/18

s) Search
p) Print
a) Analyze
q) Quit
Your selection: a

Percentage of Prime Area filled: 60%
Average nodes in overflow lists: 0.9
Longest linked list: 5

s) Search
p) Print
a) Analyze
q) Quit
Your selection: s

Please enter the item number: 731
Not found

s) Search
p) Print
a) Analyze
q) Quit
Your selection: s
```

```
Please enter the item number: 143
Price: 14

s) Search
p) Print
a) Analyze
q) Quit
Your selection: q
>
```

C-2 LEVEL 2 (25 points)

Please implement the hash table with optional hashing function, and complete the **HashTable** class of all constructors, destructor, function members, static functions, and operator overloads in below **HashTable** class declaration with the following requirements.

- ✓ The declaration of class **HashTable** is in **HashTable-2.h**.
- ✓ You only need to define the **HashTable-2.cpp.**
- ✓ The main program is in **C-2.cpp**. **DO NOT MODIFY THE MAIN PROGRAMS.**
- ✓ Do not modify and add any function and prototype to **HashTable-2.h**
- ✓ You can copy the main file, header file and testcases from /home/share/midtem/C/L2/.
- ✓ Compile

```
g++ C-2.cpp HashTable-2.cpp -o C-2
```

- ✓ **tableSize** is used to store the size of the hash table with the unit of **node**.
- ✓ **tablePtr** should be point to the dynamic memory allocated for hash table.
- ✓ **HashingFunction** should be point to the desired hashing function for hash table.
- ✓ In constructor, You have to allocate dynamic memory for hash table
- ✓ In destructor, you have to release the memory of hash table.
- ✓ **setHF(int option)** is used to set hashing function with specified option number.

```
option = 1 : Modulo-division hashing
option = 2 : Digit-extraction hashing
option = 3 : Folding hashing
option = 4 : Pseudorandom hashing
```

- ✓ **operator += (node n)** is used to insert node data to hash table.

- ✓ **`printTable()`** is used to print the whole hash table on the terminal. The "Prime Area" column shows the first inserted node in that address, while the "Overflow List" column shows the nodes inserted after the first node and linked one by one in input order. The title of "Home Addr" and "Prime Area" should be split by a tab character, while "Prime Area" and "Overflow List" should be split by two tab character. The below value of "Home Addr", "Prime Area" and "Overflow List" should also be split by two tab characters. And the item number and the price should be split by '/'.
- ✓ **`searchTable()`**, ask the user to enter the item number, search for it in hash table, then print the price on the terminal.
- ✓ **`analyzeTable()`**, analyze and print out the information of hash table. The first information is the usage in the Prime Area, the second is the average nodes number in Overflow List, and the last is the longest length of the linked list in hash table.
- ✓ **`moduloDivision(int key, int tableSize)`**, hashing function for hash table, generate the address for node data with key value and table size by modulo-division.
- ✓ **`digitExtraction(int key, int tableSize)`**, hashing function for hash table. Generate the address by extracting and merging the first digit of every two digits from the lowest digit of the key value. If the generated address is greater than the largest index of hash table, regenerate the new address by modulo the table size.

  `ex: key = 78193, tableSize = 11`

  `address = 713 > (tableSize-1) → new address = 9`
- ✓ **`Folding(int key, int tableSize)`**, hashing function for hash table. Generate the address1132866 by extracting and accumulating every two digits of the key value from the lowest digit, then discard the digits greater than two. If the generated address is greater than the largest index of hash table, regenerate the new address by modulo the table size.

  `ex: key = 78193, tableSize = 11`

  `7 + 81 + 93 = 181 → address = 81 → new address = 4`
- ✓ **`Pseudorandom(int key, int tableSize)`**, hashing function for hash table. Generate the address by pseudorandom number generator: **`y = a * x + c`**, where **`y`** is address, **`x`** is key, **`a`** and **`c`** are constants. In this problem, predefine a to 13 and c to 7.

```
#ifndef HASHTABLE_H
#define HASHTABLE_H
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <vector>
using namespace std;
struct node
{
    int itemNum;
    int price;
    node* ptr;
};
class HashTable{
private:
    int tableSize;
    node* tablePtr;
    int (*HashingFuncton)(int key, int tableSize);
public:
    HashTable(int tableSize);
    ~HashTable();

    void setHF(int option);
    void operator += (node n); //Insert data to hash table
    void printTable();
    void searchTable();
    void analyzeTable();

    static int moduloDivision(int key, int tableSize);
    static int digitExtraction(int key, int tableSize);
    static int folding(int key, int tableSize);
    static int Pseudorandom(int key, int tableSize);
};
#endif
```

You can reference **C-2.cpp** to learn about the content of the main function.

The content of **C-2.cpp**:

```
>cat C-2.cpp
#include "HashTable-2.h"
using namespace std;

int main(int argc, char** argv){
    int tableSize, option;
    ifstream in(argv[1]);
    cout << "==== Level 2: Hash Table with Opional Hashing
Function ====" << endl;
    cout << "Please enter the size of hash table: ";
    cin >> tableSize;
    HashTable HT(tableSize); //Create hash table

    //Choose hash function
    cout << "\nPlease choose a hashing function." << endl;
    cout << "1. Modulo-division" << endl;
    cout << "2. Digit-extraction" << endl;
    cout << "3. Folding" << endl;
    cout << "4. Pseudorandom" << endl;
    cout << "Your selection: ";
    cin >> option;
    HT.setHF(option);

    //Insert data to hash table
    node n;
    while(in.read((char*)&n,sizeof(n)))
        HT += n;

    //Print menu
    char sel;
    cout << "\ns) Search\np) Print\na) Analyze\nq) Quit\nYour
selection: ";
    cin >> sel;
    while(sel != 'q'){
        switch(sel){
            case 's':
                HT.searchTable();
                break;
            case 'p':
                HT.printTable();
                break;
            case 'a':
                HT.analyzeTable();
                break;
            default:
                break;
```

```
        }
        cout << "\ns) Search\np) Print\na) Analyze\nq)
 Quit\nYour selection: ";
        cin >> sel;
    }
    return 0;
 }
 >
```

The sample output of level 2:

```
>./C-2 1.dat
==== Level 2: Hash Table with Optional Hashing Function ====
Please enter the size of hash table: 10

Please choose a hashing function.
1. Modulo-division
2. Digit-extraction
3. Folding
4. Pseudorandom
Your selection: 1

s) Search
p) Print
a) Analyze
q) Quit
Your selection: p

Home Addr       Prime Area              Overflow List
0               98150/50
1
2               11252/12
3                  13033/30                    45173/17, 36133/13,
87143/14, 78193/19(continue to the upper line)
4               66484/12
5
6               15646/56                69166/66, 66176/76
7               89197/19                61167/16, 57117/11
8
9               29189/18

s) Search
p) Print
a) Analyze
q) Quit
Your selection: a

Percentage of Prime Area filled: 70%
Average nodes in overflow lists: 0.8
```

```
Longest linked list: 5

s) Search
p) Print
a) Analyze
q) Quit
Your selection: s

Please enter the item number: 87143
Price: 14

s) Search
p) Print
a) Analyze
q) Quit
Your selection: s

Please enter the item number: 10383
Not found

s) Search
p) Print
a) Analyze
q) Quit
Your selection: q
>./C-2 1.dat
==== Level 2: Hash Table with Optional Hashing Function ====
Please enter the size of hash table: 11

Please choose a hashing function.
1. Modulo-division
2. Digit-extraction
3. Folding
4. Pseudorandom
Your selection: 2

s) Search
p) Print
a) Analyze
q) Quit
Your selection: p

Home Addr       Prime Area              Overflow List
0               69166/66                57117/11, 66176/76
1               11252/12                15646/56, 61167/16
2
3               89197/19
4               13033/30
5               36133/13
```

```
6                45173/17                66484/12
7
8                98150/50
9                78193/19
10               87143/14                29189/18

s) Search
p) Print
a) Analyze
q) Quit
Your selection: q
>./C-2 1.dat
==== Level 2: Hash Table with Optional Hashing Function ====
Please enter the size of hash table: 12

Please choose a hashing function.
1. Modulo-division
2. Digit-extraction
3. Folding
4. Pseudorandom
Your selection: 3

s) Search
p) Print
a) Analyze
q) Quit
Your selection: p

Home Addr        Prime Area              Overflow List
0                89197/19                61167/16
1                36133/13
2
3                15646/56                69166/66
4                13033/30                45173/17, 98150/50
5                11252/12
6                66484/12
7                66176/76
8
9                78193/19                57117/11
10               87143/14                29189/18
11

s) Search
p) Print
a) Analyze
q) Quit
Your selection: q
>./C-2 1.dat
==== Level 2: Hash Table with Optional Hashing Function ====
```

```
Please enter the size of hash table: 12

Please choose a hashing function.
1. Modulo-division
2. Digit-extraction
3. Folding
4. Pseudorandom
Your selection: 4

s) Search
p) Print
a) Analyze
q) Quit
Your selection: p

Home Addr       Prime Area              Overflow List
0               45173/17                29189/18
1
2
3               11252/12                66176/76
4               57117/11
5               15646/56                69166/66
6               87143/14
7
8               13033/30                89197/19, 36133/13,
78193/19
9               98150/50
10              61167/16
11              66484/12

s) Search
p) Print
a) Analyze
q) Quit
Your selection: q
>
```

National Yang Ming Chiao Tung University
Department of Electrical & Computer Engineering
Computer Intelligence on Automation (C.I.A.) Lab

【PROBLEM SET D】 (100%)

- **Please make sure you check this page's ✓ (notification and hints).**

As a National Yang Ming Chiao Tung University student, you may be very familiar to using the E3 platform. E3 platform is a digital platform that promotes the interaction between the students and the teachers. In order to use the E3 platform, each of the users has to **register** their own account at first. After logging the system, you can see the course which you have selected. If you click any course button, you can **see the homework** and the material of this course. However, for a **teacher's account**, you not only can see the course's information but be able to **announce the homework** on the course page.

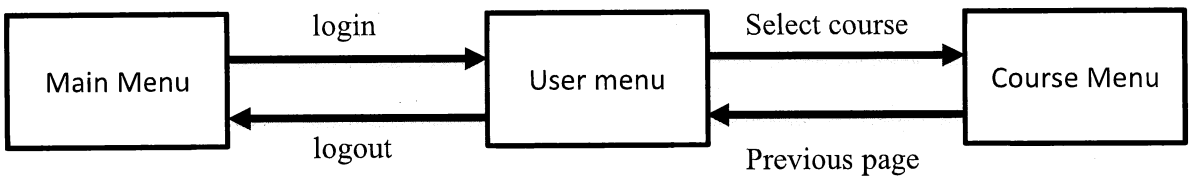
In this problem set, you are going to implement three classes, **class Course, class Teacher, class Student**.

- ✔ The declarations of these classes are all in **Class.h**.
- ✔ The main programs are in `D.cpp`. **DO NOT MODIFY THE MAIN PROGRAMS.**
- ✔ Do not modify and add any function and prototype to `Class.h`

- ✔ In this problem, you need to use the vector container. If you have any question, you can refer to the following website:
  https://shengyu7697.github.io/std-vector/
  - Declare: vector<int> num_vec; //can view as an empty integer array
  - Add element: num_vec.push_back(5); //num_vec = {5}
  - Get vector size: num_vec.size(); //num_vec.size() = 1;
  - Get/modify element: num_vec[0]; //similar to array
  - Delete the last element: num_vec.pop_back(); //num_vec = {}
- ✔ **If you want to use new to create a dynamic memory and call constructor as well, you can use the following way.**
- ✔ Ex: Course* c_ptr = new Course(name, teacher_ptr);

- ✔ You can copy D.cpp, Class.h and Class.cpp from /home/share/midtem/D/.
- ✔ Compile

```
g++ D.cpp Class.cpp -o D
```



D-1 LEVEL 1 (80 points)

Class Course:

- ✓ *tea_arr* is used for creating a dynamic array of (Teacher*) and *stu_arr* is in the same way.
- ✓ There are at most 2 teachers and 5 students in a course. (testcase will not violate)
- ✓ *total_vec* stores all of the course objects' name. *total_obj_ptr_vec* stores all of the course objects' pointer. **Don't forget to maintain them.**
- ✓ **`void createNewCourse(string name, Teacher* teacher_ptr)`**, static function, in this function, you need to construct a new Course object.
- ✓ There are two constructors to create **`Course`** objects: **`Course()`, `Course(string name, Teacher* teacher_ptr)`.** The default one can be empty. For the second constructor, there should be one teacher as the first course teacher.
- ✓ In constructors and destructor, you have to maintain the memory of dynamic array
- ✓ **`operator<<`**, output the course's information(course name, teacher array, student array, homework vector).

```
class Course
{
private:
    int stu_num;
    int teacher_num;
    string name;
    Teacher **tea_arr;
    Student **stu_arr;
    vector<string> HW_vec;

public:
    static vector<string> total_vec;
    static vector<Course*> total_obj_ptr_vec;
    static void createNewCourse(string name, Teacher* teacher_ptr);

    friend class Teacher;
    friend class Student;
    Course();
    Course(string name, Teacher* teacher_ptr);
    ~Course();

    friend ostream& operator<<(ostream& out, Course& c);
};
```

Class Teacher:

- ✓ *course_arr* is used for creating a dynamic array of (Course*) and there are at most 3 courses for a teacher. (testcase will not violate)
- ✓ *total_vec* stores all of the teachers' name. *total_obj_ptr_vec* stores all of the Teacher objects' pointer. **Don't forget to maintain them.**
- ✓ **`void createNewTeacher(string name, string password)`**, static function, in this function, you need to construct a new Teacher object.
- ✓ There are two constructors to create **`Teacher`** objects: **`Teacher()`**, **`Teacher(string name, string password)`**. The default one can be empty.
- ✓ In constructors and destructor, you have to maintain the memory of dynamic array and the static data member.
- ✓ **`check_password(string input)`**, a member function, use to return whether the input is equal to the correct password or not.
- ✓ **`menu()`**, a member function, use to show the menu and call the function to the corresponding input.
- ✓ **get_name()**, a member function, use to return the teacher's name.
- ✓ **`operator+=`**, an overloading operator. This operator means the teacher is going to join in the course.

```
class Teacher
{
private:
    int course_num;
    string name;
    Course **course_arr;
    const string password;

public:
    static vector<string> total_vec;
    static vector<Teacher*> total_obj_ptr_vec;
    static void createNewTeacher(string name, string password);

    Teacher();
    Teacher(string name, string password);
    ~Teacher();
    bool check_password(string input) const;
    bool menu();
    string get_name() const;
    void operator+=(Course *course_ptr);
};
```

Class Student:

- ✓ *course_arr* is used for creating a dynamic array of (Course*) and there are at most 3 courses for a student. (testcase will not violate)
- ✓ *total_vec* stores all of the students' name. *total_obj_ptr_vec* stores all of the student objects' pointer. **Don't forget to maintain them.**
- ✓ **`void createNewStudent(string name, string password)`**, static function, in this function, you need to construct a new Student object.
- ✓ There are two constructors to create **`Student`** objects: **`Student()`**, **`Student(string name, string password)`**. The default one can be empty.
- ✓ In constructors and destructor, you have to maintain the memory of dynamic array and the static data member.
- ✓ **`check_password(string input)`**, a member function, use to return whether the input is equal to the correct password or not.
- ✓ **`menu()`**, a member function, use to show the menu and call the function to the corresponding input.
- ✓ **`select_course()`**, a member function. In this function, you should ask user to enter a course's name, add student to that course and add the course to student's course_arr.
- ✓ **`get_name()`**, a member function, use to return the student's name.

```
class Student
{
private:
    int course_num;
    string name;
    const string password;
    Course **course_arr;
public:
    static vector<string> total_vec;
    static vector<Student*> total_obj_ptr_vec;
    static void createNewStudent(string name, string password);

    friend class Teacher;
    Student();
    Student(string name, string password);
    ~Student();
    bool check_password(string input) const;
    bool menu();
    void select_course();
    string get_name() const;
};
```

The sample output of level 1:

```
>./D

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 3

Your name: wen
password: wen

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 3

Your name: hp
password: hp

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 5

Your name: su
password: su

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 5

Your name: wu
```

```
password: wu

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 5

Your name: wang
password: wang

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 6

Course name: oop
Teacher name: wen
password: wen

Open course successfully

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 6

Course name: alg
Teacher name: hp
password: hp

Open course successfully

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
```

```
5. Student Register
6. Add a New Course
Choice: 4

Name: su
password: su

Login Success
----------User Menu----------
You have 0 courses.
1. Select new course.
2. Back to teacher mode.
3. Logout.
Choice: 2
You are not available to do this

----------User Menu----------
You have 0 courses.
1. Select new course.
2. Back to teacher mode.
3. Logout.
Choice: 1
course name: oop
----------User Menu----------
You have 1 courses.
1. oop
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice: 1
----------Course Menu----------
this course is oop
teacher: wen
student: su
homework:
1. Previous page.
Choice: 1
----------User Menu----------
You have 1 courses.
1. oop
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice: 4
Back to the Main Menu


----------Main Menu----------
1. Search a Course
```

```
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 4

Name: wu
password: wu

Login Success
----------User Menu----------
You have 0 courses.
1. Select new course.
2. Back to teacher mode.
3. Logout.
Choice: 1
course name: oop
----------User Menu----------
You have 1 courses.
1. oop
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice: 4
Back to the Main Menu


----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 4

Name: wang
password: wang

Login Success
----------User Menu----------
You have 0 courses.
1. Select new course.
2. Back to teacher mode.
3. Logout.
Choice: 1
course name: oop
----------User Menu----------
```

```
You have 1 courses.
1. oop
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice: 4
Back to the Main Menu


----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 2

Name: wen
password: wen

Login Success
----------User Menu----------

You have 1 courses.
1. oop
2. Simulate student mode.
3. Logout.
Choice: 1
----------Course Menu----------
this course is oop
teacher: wen
student: su, wu, wang
homework:
1. Assign the homework
2. Add new teacher
3. Previous page.
Choice: 1
Name: HW0
----------User Menu----------

You have 1 courses.
1. oop
2. Simulate student mode.
3. Logout.
Choice: 1
----------Course Menu----------
this course is oop
teacher: wen
```

```
student: su, wu, wang
homework: HW0
1. Assign the homework
2. Add new teacher
3. Previous page.
Choice: Lab1
Choice: 1
Name: Lab1
----------User Menu----------

You have 1 courses.
1. oop
2. Simulate student mode.
3. Logout.
Choice: 1
----------Course Menu----------
this course is oop
teacher: wen
student: su, wu, wang
homework: HW0, Lab1
1. Assign the homework
2. Add new teacher
3. Previous page.
Choice: 2
Name: hp
----------User Menu----------

You have 1 courses.
1. oop
2. Simulate student mode.
3. Logout.
Choice: 3
Back to the Main Menu


----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 1

Name: oop
this course is oop
teacher: wen, hp
student: su, wu, wang
homework: HW0, Lab1
```

```
----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 2

Name: hp
password: hp

Login Success
----------User Menu----------

You have 2 courses.
1. alg
2. oop
3. Simulate student mode.
4. Logout.
Choice: 1
----------Course Menu----------
this course is alg
teacher: hp
student:
homework:
1. Assign the homework
2. Add new teacher
3. Previous page.
Choice: 3
Back to the Main Menu

----------User Menu----------

You have 2 courses.
1. alg
2. oop
3. Simulate student mode.
4. Logout.
Choice: 4
Back to the Main Menu


----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
```

```
5. Student Register
6. Add a New Course
Choice:
```

D-2 LEVEL 2 (20 points)

Please add the following data member and function member to the corresponding class. After you finish these functions, the teacher account can simulate a student account.

Class Teacher:

- ✓ **`operator Student()`**, an overloading operator, use to return a Student object, copy from the teacher object and this student account should be simulated.
- ✓ **`Simulate_Student_mode()`**, a member function, use to create a local Student object which casts from the this object and use it to call Student's menu function.

```
class Teacher
{
public:
    operator Student() const;
     void Simulate_Student_mode();
};
```

Class Student:

- ✓ **`bool simulate`**, means this student account is simulated or not.

  simulated student account:

  - When it created, it will not be added to the Student's *total_vec* and *total_obj_ptr_vec.*
  - It can select courses. However, it will not be added to the course's student array.
  - Any action when simulated will not influence the origin teacher object.

- ✓ **`Back_to_Teacher_mode()`**, a member function, go back to the Teacher's menu function. A not simulated account is not allowed to use this function.

```
class Teacher
{
private:
    bool simulate;
public:
    void Back_to_Teacher_mode();
};
```

The sample output of level 2: (continue from the level 1's output)

```
----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 2

Name: wen
password: wen

Login Success
----------User Menu----------

You have 1 courses.
1. oop
2. Simulate student mode.
3. Logout.
Choice: 2
------------------------
You are in student mode
----------User Menu----------
You have 1 courses.
1. oop
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice: 1
----------Course Menu----------
this course is oop
teacher: wen, hp
student: su, wu, wang
homework: HW0, Lab1
1. Previous page.
Choice: 1
----------User Menu----------
You have 1 courses.
1. oop
```

```
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice: 2
course name: alg
----------User Menu----------
You have 2 courses.
1. oop
2. alg
3. Select new course.
4. Back to teacher mode.
5. Logout.
Choice: 2
----------Course Menu----------
this course is alg
teacher: hp
student:
homework:
1. Previous page.
Choice: 1
----------User Menu----------
You have 2 courses.
1. oop
2. alg
3. Select new course.
4. Back to teacher mode.
5. Logout.
Choice: 4
----------------------------
You are back to teacher mode
----------User Menu----------

You have 1 courses.
1. oop
2. Simulate student mode.
3. Logout.
Choice: 2
------------------------
You are in student mode
----------User Menu----------
You have 1 courses.
1. oop
2. Select new course.
3. Back to teacher mode.
4. Logout.
Choice:
```

Another Sample output for level 2.

```
>./D

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 3

Your name: w
password: w

----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: 2

Name: w
password: w

Login Success
----------User Menu----------

You have 0 courses.
1. Simulate student mode.
2. Logout.
Choice: 1
------------------------
You are in student mode
----------User Menu----------
You have 0 courses.
1. Select new course.
2. Back to teacher mode.
3. Logout.
Choice: 2
---------------------------
You are back to teacher mode
----------User Menu----------

You have 0 courses.
1. Simulate student mode.
2. Logout.
```

```
Choice: 1
------------------------
You are in student mode
----------User Menu----------
You have 0 courses.
1. Select new course.
2. Back to teacher mode.
3. Logout.
Choice: 3
Back to the Main Menu


----------Main Menu----------
1. Search a Course
2. Teacher Login
3. Teacher Register
4. Student Login
5. Student Register
6. Add a New Course
Choice: ^C
```

【PROBLEM SET E】 (60%)

The interaction of different class objects is a very commonly used function in OOP. In this problem set, you need to create three classes called Item, Factory and Store. These classes are used to create three objects to complete the operation of store and factory.

- ✔ You need to complete each cpp file to make E.cpp work.
- ✔ The main program is in **E.cpp**. **DO NOT MODIFY THE MAIN PROGRAMS.**
- ✔ You can copy all the files you may need from /home/share/midterm/E/.
- ✔ Compile

  Level 1 (20%):

  Demo Part A. Factory mode function.

  Level 2 (40%):

  Demo Part B. Store mode function.

E-1 LEVEL 1 (20 points)

Level 1 is to build a factory system. The factory will manufacture items according to the production list. Each item has the following parameters.

```
class Item
{
private:
    std::string name;//item name
    int quantity;     //quantity
    double cost;      //manufacturing cost(set in factory mode)
    double price;     //commodity price(set in store mode)
    struct Date{
    int year;
    int month;
    int date;
    }EXP;             //expiration date(yyyy/mm/dd)
};
```

In the main function, the system will ask the user to input 'A' to enter the factory mode. And there are two functions in Factory Mode.

1. Manufacturing products: In this function, the system will ask the user to input factory filename to get the production list. And you need to store these items' data by your way.
2. Show Factory Product: This function is to display the list according to the expiration date of the product (or alphabetically when the expiration date is the same). You should use overload operator << to complete this function.

```
$ cat factory1.in
apple 5 2.5 2023/04/17
banana 10 1.2 2023/04/10
apple 7 3 2023/04/19
orange 12 2 2023/04/18
grape 3 7 2023/04/18
```

✓ The order of data is name, quantity, cost, and expired date.

```
$ ./E
Choose A or B
A. Factory Mode
B. Store Mode
A
Input 1~2 for choose function
1. Manufacturing products
2. Show Factory Product
1
input factory manufacture list filename
factory1.in
Choose A or B
A. Factory Mode
B. Store Mode
A
Input 1~2 for choose function
1. Manufacturing products
2. Show Factory Product
2
Factory Stock list:
banana - Quantity: 10 - cost $1.2 EXP:2023/4/10
apple  - Quantity:  5 - cost $2.5 EXP:2023/4/17
grape  - Quantity:  3 - cost $  7 EXP:2023/4/18
orange - Quantity: 12 - cost $  2 EXP:2023/4/18
apple  - Quantity:  7 - cost $  3 EXP:2023/4/19
```

✓ Please follow the format of the output.

E-2 LEVEL 2 (40 points)

Level 2 is Store Mode. The store will purchase goods from the factory and make profits according to the price difference in the middle, but at the same time, it will also face losses due to expired goods.

In Store Mode, there are four functions to achieve the above functions.

1. Purchase commodity: After entering the store filename, it will purchase goods from the factory according to the information inside. Note that the list of goods in the factory will also change at the same time. If the product quantity is 0, delete this product.
2. Settlement profit: The system will ask you to enter the date (yyyy/mm/dd) and the name of the sale file. First, the expired items in the store list will be cleared, and these costs will be recorded in the variable **loss**, and then the net profit will be calculated according to the sales quantity in the file. Note that the products will be sold in order according to the expiration date.
3. Show store menu: This function will list the item list of the store in subtitle order. Note that the same product name will only appear once.
4. Show commodity list: Calling this function will list the items in the store according to the expiration date, helping users understand the store's inventory.

```
$ cat store1.in
apple 10 4
banana 8 1.5
orange 15 2.5
grape 4 9
$ cat sale1.in
apple 7
grape 1
orange 6
```

System output of Store mode (Use the above factory product):

```
Choose A or B
A. Factory Mode
B. Store Mode
B
Input 1~4 for choose function
1. Purchase commodity
2. Settlement profit
3. Show menu
4. Show commodity list
1
input store purchase list filename
store1.in
orange : no enough product
grape : no enough product
lemon : no this product
Choose A or B
A. Factory Mode
B. Store Mode
B
Input 1~4 for choose function
1. Purchase commodity
2. Settlement profit
3. Show menu
4. Show commodity list
4
commodity list:
banana - Quantity:  8 - price $1.5 cost:1.2  EXP:2023/4/10
apple  - Quantity:  5 - price $  4 cost:2.5  EXP:2023/4/17
grape  - Quantity:  3 - price $  9 cost:  7  EXP:2023/4/18
orange - Quantity: 12 - price $2.5 cost:  2  EXP:2023/4/18
apple  - Quantity:  5 - price $  4 cost:  3  EXP:2023/4/19
Choose A or B
A. Factory Mode
B. Store Mode
B
Input 1~4 for choose function
1. Purchase commodity
2. Settlement profit
3. Show menu
4. Show commodity list
3
Store menu:
apple  - Quantity: 10 - price $  4
banana - Quantity:  8 - price $1.5
grape  - Quantity:  3 - price $  9
orange - Quantity: 12 - price $2.5
Choose A or B
A. Factory Mode
```

```
B. Store Mode
A
Input 1~2 for choose function
1. Manufacturing products
2. Show Factory Product
2
Factory Stock list:
banana - Quantity: 2 - cost $1.2 EXP:2023/4/10
apple  - Quantity: 2 - cost $  3 EXP:2023/4/19
Choose A or B
A. Factory Mode
B. Store Mode
B
Input 1~4 for choose function
1. Purchase commodity
2. Settlement profit
3. Show menu
4. Show commodity list
2
input the date yyyy/mm/dd and sale list filename
2023/04/12 sale1.in
2023/04/12 profit is: 4.9
Choose A or B
A. Factory Mode
B. Store Mode
B
Input 1~4 for choose function
1. Purchase commodity
2. Settlement profit
3. Show menu
4. Show commodity list
4
commodity list:
grape  - Quantity: 2 - price $  9 cost:7  EXP:2023/4/18
orange - Quantity: 6 - price $2.5 cost:2  EXP:2023/4/18
apple  - Quantity: 3 - price $  4 cost:3  EXP:2023/4/19
Choose A or B
A. Factory Mode
B. Store Mode
B
Input 1~4 for choose function
1. Purchase commodity
2. Settlement profit
3. Show menu
4. Show commodity list
3
Store menu:
apple  - Quantity: 3 - price $  4
grape  - Quantity: 2 - price $  9
```

```
orange - Quantity: 6 - price $2.5
```

【PROBLEM SET F】 (60%)

ATPG stands for Automatic Test Pattern Generation. It is a process used in digital circuit design to create test patterns that can be used to test the circuit's functionality and to detect any faults or defects in the circuit. The ATPG process involves the use of algorithms and software tools to generate a set of test patterns that can be applied to the circuit to check its performance.

In this problem set, you need to build the basic ATPG structure with two classes, including Netlist and Gate.

- ✔ The class **`Netlist`** and class **`Gate`** are declared in a nested structure in **`Netlist.h`**.
- ✔ You need to define two classes in **`Netlist.cpp`**.
- ✔ The main programs are in **`F-1.cpp, F-2.cpp, F-3.cpp`**. **DO NOT MODIFY THE MAIN PROGRAMS.**
- ✔ Do not modify and add any function and prototype to **`Netlist.h`**.
- ✔ To pass the test, your program cannot contain memory leaks. You can use the following command to test for memory leaks.

  ```
  > valgrind <your_executable_file> <arguments_if_needed>
  ex: valgrind ./F-1 netlist1.txt
  ```

  Valgrind reference:
  https://web.stanford.edu/class/archive/cs/cs107/cs107.1234/resources/valgrind.html
- ✔ You can copy three main files, header files, and test cases from /home/share/midtem/F/.
- ✔ Compile

  Level 1 (20%):
  ```
  g++ F-1.cpp Netlist.cpp -o ./F-1
  ```
  Level 2 (20%):
  ```
  g++ F-2.cpp Netlist.cpp -o ./F-2
  ```
  Level 3 (20%):
  ```
  g++ F-3.cpp Netlist.cpp -o ./F-3
  ```

REFERENCE

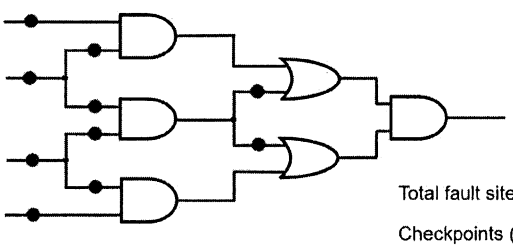
How to use vector: (you can focus on size(), push_back(), and pop_back())
https://cplusplus.com/reference/vector/vector/

F-1 LEVEL 1 – BASIC CONSTRUCTION (20 points)

Please complete the **`Netlist`** and **`Gate`** class of all constructors, destructor, accessors, mutators, and functions in below declaration with the following requirements.

`Netlist Part`

- ✓ Build constructor and destructor, remember to **free the memory** in destructor.
- ✓ **`Initialize(string inputFile)`**, use the gate data in the input file to construct the netlist.
- ✓ **`PrintInfo()`**, print out the netlist information, including #Gate, #PI, #PO, maximum fanout number, list of gates that have maximum fanout, the fault site before and after applying checkpoint theorem, and the whole netlist .
- ✓ **`No_Gate()`**, return the size of Gatelist.
- ✓ **`No_PI()`**, return the size of PIlist.
- ✓ **`No_PO()`**, return the size of POlist.
- ✓ **`FindMaxFanout()`**, find the maximum fanout number, and all the gate members that have maximum fanout, should be called in **`PrintInfo()`**.
  - fanout: the number of inputs that a particular output is connected to.
- ✓ **`CheckpointThm()`**, compute the fault site before and after applying checkpoint theorem, should be called in **`PrintInfo()`**.
  - Checkpoint theorem: A test set that detects all single stuck-at faults on all checkpoints of a combinational circuit, also detects all single stuck-at faults in that circuit. In this way, the time and resources required for fault simulation can be significantly reduced, allowing designers to test and debug their circuits more efficiently.
  - **Primary inputs** and **fanout branches** of a combinational circuit are called checkpoints.
  - **Example**



Total fault sites = 16, Total fault = 16*2 = 32

Checkpoints (•) = 10, Collapsed fault = 10*2 = 20

- **Reference:**
  http://tiger.ee.nctu.edu.tw/course/Testing2020/notes/pdf/ch3.fault_modeling.pdf

Gate Part

- ✓ Build constructor and destructor.
- ✓ **`void AddInput_list(Gate *gptr)`** , add a gate into its input_list (fanin).
- ✓ **`void AddOutput_list(Gate *gptr)`** , add a gate into its output_list (fanout).
- ✓ **`GetName()`**, an accessor function, use to read from private data members. Return the name of this gate.
- ✓ **`GetType()`**, an accessor function, use to read from private data members. Return the type of this gate.
  - Type: PI, PO, NOT, NOR, NAND
- ✓ **`GetOutput_list_Size()`**, an accessor function, use to read from private data members. Return the output_list size of this gate.
  - Type: INPUT, OUTPUT, NOT, NOR, NAND

```
Gate Format in netlist.txt
<Gate Type> <IN1> <IN2> <OUT>
Ex. NAND G1 G2 net4
● INPUT only has fanout, OUTPUT only has fanin
● NOT only has single fanin
```

```
#ifndef NETLIST_H
#define NETLIST_H
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;

class Netlist
{
    friend class Gate;
private:
    class Gate
    {
    private:
        string Name;
        string Type;
        int Value;
        int Delay;
        vector<Gate *> Input_list;
        vector<Gate *> Output_list;

    public:
        // Level 1
        Gate(string Name, string Type);
        ~Gate();

        void AddInput_list(Gate *gptr);
        void AddOutput_list(Gate *gptr);
        string GetName();
        string GetType();
        int GetOutput_list_Size();
        // Level 2
        int GetValue();
        int GetDelay();
        // Level 3
        void SetValue(int);
        void SetDelay(int);
        vector<Gate *> &GetInput_list() { return Input_list; }
        vector<Gate *> &GetOutput_list() { return Output_list; }
    };

    vector<Gate *> Gatelist;
    vector<Gate *> PIlist;
    vector<Gate *> POlist;
```

```
// Level 1
    void FanOutList();
    int No_Gate();
    int No_PI();
    int No_PO();
    void FindMaxFanout();
    void CheckpointThm();
    // Level 2
    void SetNodeValue(Gate *gptr);

public:
    // Level 1
    Netlist();
    ~Netlist();
    void Initialize(string inputFile);
    void PrintInfo();
    // Level 2
    void Simulate(string inputFile);
    void DelayAssignment(string inputFile);
    // Level 3
    void Traversal(string PI, string PO);
    int Trace(Gate *gptr, string PI, int &pathCount,
vector<string> &path);
};

#endif
```

You can reference **F-1.cpp** to learn about the content of the main function.

The content of **F-1.cpp**:

```
>cat F-1.cpp
#include <string>
#include <vector>
#include <fstream>
#include "Netlist.h"
using namespace std;
int main(int argc, char *argv[])
{
    string in_file = argv[1];
    Netlist n1;
    n1.Initialize(in_file);
    n1.PrintInfo();
    return 0;
}
>
```

The sample output of level 1: (Fanin, Fanout output in ascii order.)

```
>./F-1 netlist1.txt
# Gate: 14
# PI: 5
# PO: 2
Max Fanout Number: 2
Max Fanout List:
- G2    Fanout: n60 net18
- G3    Fanout: net14 net17
- net14 Fanout: net18 net25
Total fault: 36
Checkpoint Thm: 22
Reduce Fault to: 61.11%

//// Netlist Info ////
INPUT   G1      Fanout: net17
INPUT   G2      Fanout: n60 net18
INPUT   G3      Fanout: net14 net17
INPUT   G4      Fanout: net14
INPUT   G5      Fanout: n60
OUTPUT  G16     Fanin: G16
OUTPUT  G17     Fanin: G17
NOR     n60     Fanin: G2 G5    Fanout: G17
NAND    net14   Fanin: G3 G4    Fanout: net18 net25
NAND    net17   Fanin: G1 G3    Fanout: G16
NOT     net25   Fanin: net14    Fanout: G17
NAND    net18   Fanin: G2 net14         Fanout: G16
NOR     G17     Fanin: n60 net25        Fanout: G17
NAND    G16     Fanin: net17 net18      Fanout: G16
>
```

F-2 LEVEL 2 - SIMULATION (20 points)

Please complete the following function in **Netlist** class to implement simulation.

- ✓ **Simulate(string intput_file)**, read the input pattern (Binary file), do the simulation and output the final result.
- ✓ **SetNodeValue (Gate *gptr)**, set the value of gate according to its fanin and gate type.
- ✓ **GetValue()**, an accessor function, use to read from private data members. Return the value of this gate.
- ✓ **SetValue(int)**, a mutator function, use to write private data members. Set the value of this gate.
- ✓ You can reference **F-2.cpp** to learn about the content of the main function.

The content of **F-2.cpp**:

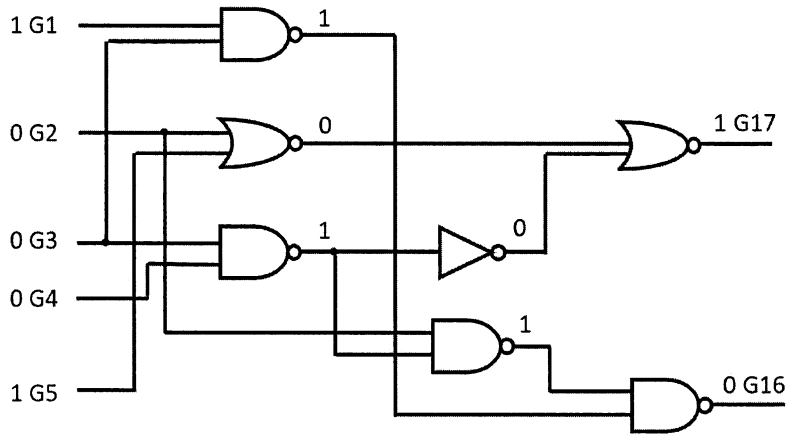
```
>cat F-2.cpp
#include <string>
#include <vector>
#include <fstream>
#include "Netlist.h"
using namespace std;

int main(int argc, char *argv[])
{
    string in_file = argv[1];
    string PI_pattern = argv[2];
    Netlist n1;
    n1.Initialize(in_file);
    n1.Simulate(PI_pattern);
    return 0;
}
>
```

The sample output of level 2: (output please follow PO order in netlist.txt)

```
>./F-2 netlist1.txt 1.dat
Logic simulation
11010 11
11000 11
10001 01
11010 11
00101 01
00100 00
10010 00
01111 00
10010 00
```

```
11011 11
>
```

Visualization:



F-3 LEVEL 3 - PATH TRAVERSAL (20 points)

Please complete the following function in **Netlist** class to implement traversal flow.

- ✓ **DelayAssignment(string intput_file)**, read the Gate_Delay File, assign the delay for gates in the netlist.
- ✓ **Traversal(string PI, string PO)**, given starting PI and ending PO, please list and count all possible paths connecting the given PI and PO. Also, print out the max propagation delay.
  - ■ The test circuit will only be combinational circuits, so there's no loop in the circuit.
- ✓ **Trace(Gate *gptr, string PI, int &pathCount, vector<string> &path)**, a function help you implement the traversal, you can use pathCount to record the number of path, use path vector to store the gates you've visited, and the return int value will be the max delay from PI to this gate.
- ✓ **GetDelay()**, an accessor function, use to read from private data members. Return the delay of this gate.
- ✓ **SetDelay(int)**, a mutator function, use to write private data members. Set the delay of this gate.
- ✓ You can reference **F-3.cpp** to learn about the content of the main function.

The content of **F-3.cpp**:

```
>cat F-3.cpp
#include <string>
#include <vector>
#include <fstream>
#include "Netlist.h"
using namespace std;

int main(int argc, char *argv[])
{
    string in_file = argv[1];
    string delay_file = argv[2];
    Netlist n1;
    n1.Initialize(in_file);
    n1.DelayAssignment(delay_file);
    string PI, PO;
    cin >> PI >> PO;
    n1.Traversal(PI, PO);

    return 0;
}
>
```

The sample output of level 3:

```
>./F-3 netlist1.txt delay1.txt
PI: G3
PO: PO_G16
Path traversal
G3 net17 G16 PO_G16
G3 net14 net18 G16 PO_G16
The paths from G3 to PO_G16: 2
Max propagation delay: 9
>
```

Visualization: (we can see there are 2 paths from G3 to PO_G16, and the longest one has propagation delay = 4 + 3 + 2 = 9)