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

UEE1303(1009) S22

Midterm Examination

FULL SCORES:
120 %

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

INSTRUCTIONS:

This midterm examination is composed of 5 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(1009) S22: Midterm Examination Demo Sheet

Student ID # : _____, Name :_____

Full Scores: up to 120 points

→You may pick arbitrary numbers of problems to solve.

Problem Set A (40%)		Problem Set B (50%)	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (10%)		Q1 (15%)	
Q2 (15%)		Q2 (15%)	
Q3 (15%)		Q3 (20%)	
Problem Set C (50%)		**Problem Set D (60%)**	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (10%)		Q1 (15%)	
Q2 (15%)		Q2 (25%)	
Q3 (25%)		Q3 (20%)	

Total Score: _____ / 120

【PROBLEM SET A】(40%)

In this problem, you should implement a class with a given struct. The class is **`Point_Set`** and the given struct is **`Point`**.

Functions and variables of the class you should add are illustrated in the main function.

Class **`Point_Set`** and struct **`Point`** are described in **Point_Set.h**.
There are three questions in this problem. For each question, we supply a main function. For A01 is main_A01.cpp. For A02 is main_A02.cpp. For A03 is main_A03.cpp.

These files are put in ***/home/share/mid/A***.

You may add code in Point_Set.h, but do not modify the existing code.

<Compile>
A-01 (10%):
$ **g++ Point_Set.cpp main_A01.cpp -o main_A01**
A-02 (15%):
$ **g++ Point_Set.cpp main_A02.cpp -o main_A02**
A-03 (15%):
$ **g++ Point_Set.cpp main_A03.cpp -o main_A03**

The content of **`Point_Set.h`** as shown below:

```
#ifndef _POINT_SET_H_
#define _POINT_SET_H_
#include <string>
using namespace std;

struct Point
{
    double x = 0;
    double y = 0;
    string name = "";
};

class Point_Set
{
private:
    int num;
    Point *points;
    string name;

public:
    // A01
    Point_Set();
    ~Point_Set();
```

```
    void Parser(string);
    friend void DisplayPointSet(const Point_Set &);

    // A02
    void operator+=(const double &);
    void operator*=(const double &);

    // A03
    void fit(double &, double &);
    double* predict(const double &, const double &);
};
#endif
```

A-01 (10%)

Firstly, you should complete the constructor and the destructor which have been declared in the header file. Next on, complete the `Parser` function that can parse the point set from the input file and the `DisplayPointSet` function. About the `DisplayPointSet` function, you should output points in ascending order by the magnitude (i.e., the distance from the point to the origin.) of the point. There are no points with the same magnitude in all test cases. The command line would be **$ ./main input1.in input2.in input3.in**. There is an example below.

The input file contains four kinds of information: the name of the point set, the number of points of the point set, the name of each point, and the coordinate of each point.

Input file: **input1.in**

```
point_set1: // the name of the point set
5 // the number of points
p1 (3,5) // the name of the point & the coordinate of the point
p6 (5,8)
p10 (9,-5)
p4 (9,6)
p12 (-9,-7.54)
```

Example: **main_A01.cpp**

```
#include "Point_Set.h"
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
    if (argc != 4)
    {
        cout << "Usage: ./main_A01 <input_file_1> <input_file_2>
<input_file_3>" << endl;
        return -1;
    }
```

```
    string file1, file2, file3;
    file1 = argv[1];
    file2 = argv[2];
    file3 = argv[3];
    Point_Set ps1, ps2, ps3;
    ps1.Parser(file1);
    ps2.Parser(file2);
    ps3.Parser(file3);
    DisplayPointSet(ps1);
    cout << endl;
    DisplayPointSet(ps2);
    cout << endl;
    DisplayPointSet(ps3);

    return 0;
}
```

Example output:

```
$ ./main_A01 input1.in input2.in input3.in
point_set1:
p1: (3,5)
p6: (5,8)
p10: (9,-5)
p4: (9,6)
p12: (-9,-7.54)

point_set2:
p3: (-3,-7)
p2: (5,-6.6)
p9: (5,-8)
p10: (9,-5)
p7: (-6,-9)

point_set3:
p2: (5,-6.6)
p8: (-8.1,2.7)
p14: (6.3,5.8)
p5: (11,-3.3)
p12: (-9,-7.54)
```

A-02 (15%)

In this part, the overloading operator will be implemented by yourself. The definition of the overloading operator is given as follows.

`void Point_Set::operator +=(const double &)`:

Overwrite points of object `Point_Set` after shifting each point by const double variable.

`void Point_Set::operator *=(const double &)`:

Overwrite points of object `Point_Set` after scaling each point by const double variable.

Example: **main_A02.cpp**

```
#include "Point_Set.h"
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
    if (argc != 3)
    {
        cout << "Usage: ./main_A02 <input_file_1> <input_file_2>" <<
endl;
        return -1;
    }
    string file1, file2;
    file1 = argv[1];
    file2 = argv[2];
    Point_Set ps1, ps2;
    ps1.Parser(file1);
    ps2.Parser(file2);
    DisplayPointSet(ps1);
    cout << endl;
    DisplayPointSet(ps2);
    cout << endl;
    ps1 += 2.6;
    DisplayPointSet(ps1);
    cout << endl;
    ps2 *= 1.2;
    DisplayPointSet(ps2);

    return 0;
}
```

Example output:

```
$ ./main_A02 input1.in input2.in
point_set1:
p1: (3,5)
p6: (5,8)
p10: (9,-5)
p4: (9,6)
p12: (-9,-7.54)
```

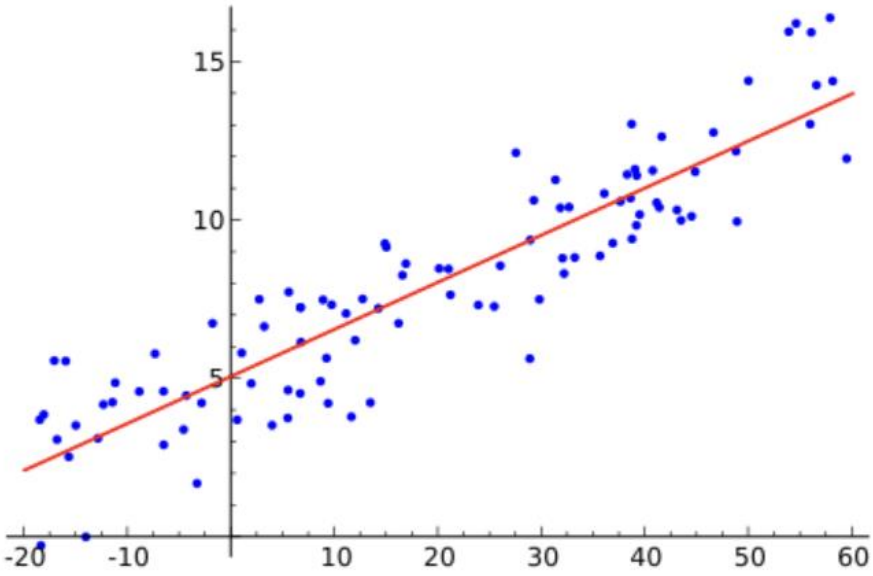
```
point_set2:
p3: (-3,-7)
p2: (5,-6.6)
p9: (5,-8)
p10: (9,-5)
p7: (-6,-9)

point_set1:
p12: (-6.4,-4.94)
p1: (5.6,7.6)
p10: (11.6,-2.4)
p6: (7.6,10.6)
p4: (11.6,8.6)

point_set2:
p3: (-3.6,-8.4)
p2: (6,-7.92)
p9: (6,-9.6)
p10: (10.8,-6)
p7: (-7.2,-10.8)
```

A-03 (15%)

Linear regression models the relation between an explanatory (independent) variable and a scalar response (dependent) variable by fitting a linear equation. For example, modeling the weights of individuals with their heights using a linear equation.



A linear regression line has an equation of the form $\boldsymbol{Y = a + bX}$, where $\boldsymbol{X}$ is the explanatory variable and $\boldsymbol{Y}$ is the dependent variable. The slope of the line is $\boldsymbol{b}$, and $\boldsymbol{a}$ is the intercept (the value of $\boldsymbol{y}$ when $\boldsymbol{x} = 0$). Without going into details, the equations that you should use are:

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \qquad \beta_1 = \frac{SS_{xy}}{SS_{xx}}$$

$$SS_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} {x_i}^2 - n(\bar{x})^2$$

$$SS_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

n: the number of points
$\bar{x}$: the average of x-coordinate value of each point
$\bar{y}$: the average of y-coordinate value of each point

In `fit` function, you should calculate coefficients based on the information of the object. For generating β_0 and β_1 at one time, we use two parameters which are passed by reference, so the initial values of the parameters taken by the `fit` function don't matter. Besides, it prints the values of β_0 and β_1 after calculating.

In `predict` function, it takes two parameters which are β_0 and β_1 we calculated, and it can calculate each point's y-coordinate value in a Point_Set by taking each point's x-coordinate value into calculation. The equation for predicting each point's y-coordinate value is given by $\boldsymbol{Y} = \beta_0 + \beta_1 \boldsymbol{X}$. Besides, you have to print each point's x-coordinate value and y-coordinate value after linear regression in the Point_Set.

Example: **main_A03.cpp**

```
#include "Point_Set.h"
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
    if (argc != 2)
    {
        cout << "Usage: ./main_A03 <input_file_1>" << endl;
        return -1;
    }
    double b_0, b_1;
    string file1;
    file1 = argv[1];
    Point_Set ps1;
    ps1.Parser(file1);
    ps1.fit(b_1, b_0);
    cout << endl;
    ps1.predict(b_1, b_0);

    return 0;
```

```
}
```

Example output:

```
$ ./main_A03 input1.in
b_0 = -0.412591
b_1 = 0.50135

Predicted Data:
(3,1.09146)
(5,2.09416)
(9,4.09956)
(9,4.09956)
(-9,-4.92474)
```

【PROBLEM SET B】(50%)

Assume you're a software engineer who's been tasked by the University Admissions Committee, or UAC, with developing a system. There are several functions in this distribution system that can help distribute the 110th high school students to their desired universities based on their wish lists.

In this problem, you should implement two different classes, one is **`Applicant`**, the other is **`University`** and you need to use TA's pb02.cpp file to complete this program please do not add new function.

- class **`Applicant`** need to be described in **Applicant.h**
- class **`University`** need to be described in **University.h**

This problem is divided into three parts, Q1, Q2, and Q3. All supplementary files will be placed in ***/home/share/mid/B/.***

<Compile>

$ **g++ -std=c++11 main.cpp Applicant.cpp University.cpp**

INPUTS

```
$ ./pb02 mid_B_1.in mid_B_1.out
$ cat mid_B_1.in ↲
U: 0001, NTU_EE, (4 X 4 4 X), (2.5 0 2.5 2.5 0), 5
U: 0002, NCKU_EE, (3 X 2 3 X), (3 3 3 3 3), 5
A: 0001, John Wang, (15 15 15 X 15), (6 3 4 5 2)
A: 0002, John Wu, (10 10 10 15 10), (1 6)
```

- There are m (m <= 100000) applicants (A_1, A_2, ..., A_m) for applying n (n <= 100000) universities (U_1, U_2, ..., U_n).

- Each U_i has five components.
  1. a serial number of this university
  2. name of this university
  3. criteria of student scores
     - (X 4 4 X 4) indicates the criteria in subjects (Math, Chinese, English, Science, Society).
     - The number in criteria are integers in the range [0, 4] which stand for [pr12, pr25, pr50, pr75, pr88].
     - For example fig_b1 shows [0, 1, 2, 3, 4] in Chinese stands for [8, 9, 11, 12, 13], and criteria (4 X 4 3 X) shows that all students who want to enter this university should meet scores [Math >= 11, X, English >= 13, Science >= 12, X].
  4. weight on student score

- (2.5 0 2.5 2.5 0) indicates the weights of the subjects (Math, Chinese, English, Science, Society)
- For example, (2.5 0 2.5 2.5 0) shows that all students who want to enter this university should compare their weighted scores: $= 2.5 \times Math + 2.5 \times English + 2.5 \times Science$.

5. enrollment limit that represents the maximum number of students

110學年度學科能力測驗
各科成績標準一覽表

標準 / 項目	頂標	前標	均標	後標	底標
國文	13	12	11	9	8
英文	13	12	8	5	4
數學	11	9	6	4	3
社會	13	12	10	8	7
自然	13	12	8	6	5

▲ fig_b1

- Each Ai has four components.
  1. a serial number of this student
  2. name of this student
  3. the student's scores in five subjects
     - (15 15 15 15 X) indicates the scores in subjects (Math, Chinese, English, Science, Society)
     - 'X' means that this student does not have a score in this subject, be careful that it is different from 0
  4. wishlist of at most six universities of this student, and the priority of the leftmost is the highest

```
$ cat main.cpp ↵
#include <iostream>
#include <cstring>
#include <fstream>

#include "Applicant.h"
#include "University.h"

using namespace std;
// Q1
void read_file(string filename, Applicant** apps, University** unis);
// Q1
```

```
void sort_apps(Applicant* apps);
// Q1
void sort_unis(University* unis);

// Q3 You need to delete the memory in the write file
void write_file(string filename, Applicant** apps, University** unis);

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

    Applicant* my_applicants;
    University* my_universities;

    if(argc < 2){
        cout << “No input file provided.” << endl;
        return -1;
    }
    if(argc < 3){
        cout << “No output file provided.” << endl;
        return -1;
    }

    //Q1
    read_file(argv[1], &my_applicants, &my_universities);
    sort_apps(my_applicants);
    sort_unis(my_universities);

    cout << my_applicants -> n_applicants << endl;
    for(int i = 0; i < my_applicants -> n_applicants; i++){
        cout << my_applicants[i] << endl;
    }

    cout << my_universities -> n_universities << endl;
    for(int i = 0; i < my_universities -> n_universities; i++){
        cout << my_universities[i] << endl;
    }

    my_applicants->get_Chinese_Avg();
    my_applicants->get_Math_Avg();
    my_applicants->get_Science_Highest();
    my_applicants->get_English_Highest();

    //Q2
    for(int i = 0; i < my_universities -> n_universities; i++){
        my_universities[i].get_Rank5(my_applicants);
    }

    //Q3
    match_func(my_applicants, my_universities);
    for(int i = 0; i < my_applicants -> n_applicants; i++){
```

```
        my_applicants[i].match_uni();
    }

    for(int i = 0; i < my_universities -> n_universities; i++){
        my_universities[i].match_apps();
    }
    write_file(argv[2], &my_applicants, &my_universities);
}
```

B-01 (15 points)

In this part, please finish the function `read_file, sort_unis, sort_apps` and two classes, remembering that you cannot change anything in the file except function `read_file, sort_unis, sort_apps` and `write_file` in the file main.cpp. It's important to keep in mind that you must sort the applicants and universities in descending order.

```
$ ./pb02 mid_B_1.in mid_B_1.out ↵
_Applicant
Number 0001
Name John Wang
Math 15
Chinese 15
English 15
Science 15
Total: 60/60

_Applicant
Number 0002
Name John Wu
Math 10
Chinese 10
English 10
Science 15
Society 10
Total: 55/75

2
_University
Number 0001
Name NTU_EE
Enrollment Limit 5

_University
Number 0005
Name NCKU_EE
Enrollment Limit 5
```

```
Chinese Average: 12.5
Math Average: 12.5
Science Highest: 15
English Highest: 15
```

B-02 (15 points)

In this part, please finish the function `get_Rank5` in the class `University` to get the serial number of the first five applicants (if the number is less than five, just print all of them) that the university wants the most based on the specific criteria and weights provided by each university; you do not need to consider the wish list of each applicant in this section.

```
$ ./pb02 mid_B_1.in mid_B_1.out ↲
_University 0001 NTU_EE: 0002
_University 0002 NCKU_EE: 0001 0002
```

In this example, 0001 fail to pass the criteria of university 0001.

B-03 (20 points)

To get the results of each applicant and university, complete the functions `write_file`, `match_func`, `match_uni` and `match_apps` in class `University` and `Applicant` to get the results of each applicant and university.

Note that if there are multiple applicants with the same weighted score, choose the applicant with the smaller serial number until the enrollment limit is reached.

```
$ ./mid_B mid_B_1.in mid_B_1.out ↲
$ cat mid_B_1.out  ↲
_Applicant
Number 0001
Name John Wang
Match Result NCKU_EE

_Applicant
Number 0002
Name John Wu
Match Result NCKU_EE

_University
Number 0001
Name NCKU_EE
Match Result

_University
```

```
Number 0002
Name NCKU_EE
Match Result 0001, 0002
```

【PROBLEM SET C】(50%)

Digital signal processing (DSP) is the use of digital processing, such as by computers or more specialized digital signal processors, to perform a wide variety of signal processing operations. The digital signals processed in this manner are a sequence of numbers that represent samples of a continuous variable in a domain such as time, space, or frequency. In this problem, you are asked to implement some operations on discrete-time signals, and at last, you'll need to implement a median filter.

The source code (**main.cpp, Seq.cpp, Seq.h**) and input files (**1.dat, 2.dat**) will be given. You only need to complete **Seq.cpp.** You may add codes in **Seq.h**, but do not modify the existing code. These codes are located in ***$ /home/share/mid/C*** and you can copy them to your workstation. Make sure to copy all of the files or the Makefile won't work successfully.

<Compile>
for C-01: $ make q1
for C-02: $ make q2
for C-03: $ make q3
<Execute>
for C-01: $ make t1
for C-02: $ make t2
for C-03: $ make t3
<Clean object code or executable files>
$ make clean

Header file**: Seq.h**

```
#ifndef _Seq_H__
#define _Seq_H__
#include <iostream>
#include <string>
class Seq
{
private:
    int *data;
    int size;
public:
    // Q-1
    Seq(std::string);
    Seq(int);
    Seq(int *data, int size);
    ~Seq();
    void display();
    //Q-2
    void operator >> (int); // right circular shift
```

```
    void operator << (int); // left circular shift
    Seq operator - (); // circular time reversal
    Seq(const Seq&); // copy constructor
    //Q-3
    friend Seq operator * (const Seq&, const Seq&); // convolution sum
    Seq mid_filter(int); // median filter
};
#endif
```

C-01 (10%)

In C-01, you need to complete all of the constructors, a destructor, and a member function `display`. There are three constructors for initializing an object. The first one is used to initialize an object with a binary file (either 1.dat or 2.dat). For understanding, the input file format is shown in decimal as below, but it's stored in binary in fact.

Input file: **1.dat**

```
5 -1 2 0 -2 1
```

The first integer denotes the sequence length, following that are the values of the discrete-time signals in this sequence. The second constructor is used when only given the sequence length. The third constructor is used when both the sequence data and the sequence size are given. The destructor is used to free the memory space that an object allocated when the object is out of its living scope. The member function `display` shows the content within a sequence in decimal. The example output is shown as the following.

Example:

```
#include "Seq.h"
using namespace std;

int main() {
    Seq seq1("1.dat");
    seq1.display();
    Seq seq2("2.dat");
    seq2.display();
    return 0;
}
```

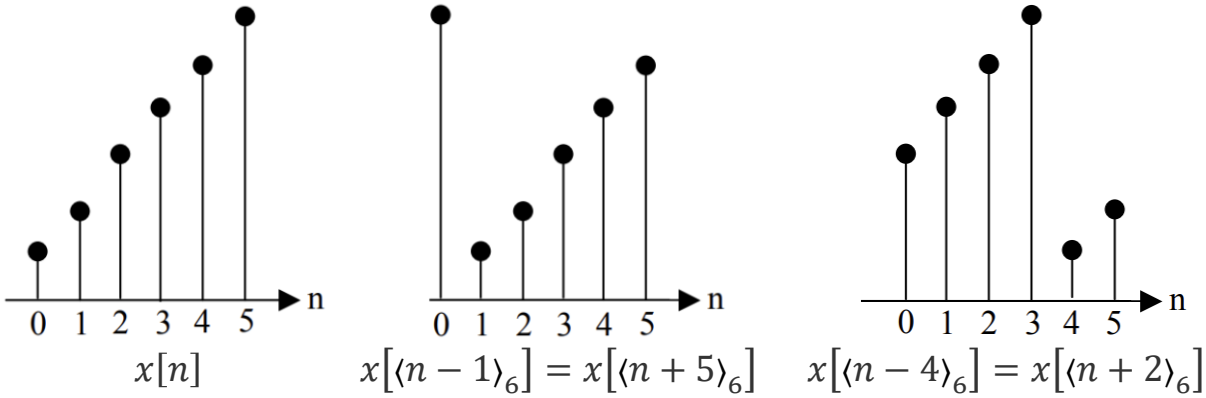
Example output:

```
-1 2 0 -2 1
1 0 -1 0 1 0
```

C-02 (15%)

In C-02, you need to complete a copy constructor and three discrete-time signal operations which are right circular shift, left circular shift and circular time reversal. Circular time-shifting operations are achieved by using the modulo operation. The circular shift of a length-N sequence x[n] by an amount n_0 sample periods is defined by the equation $x_c[n] = x[\langle n - n_0 \rangle_N]$, where $x_c[n]$ is also a length-N sequence. If $n_0 > 0$, it is a right circular shift, and if $n_0 < 0$, it is a left circular shift. For $n_0 > 0$, the above equation implies $x_c[n] = \{\ x[n - n_0],\ for\ n_0 \le n \le N-1\quad x[N - n_0 + n],\ for\ 0 \le n \le n_0$

The concept of a circular shift of a finite-length sequence is illustrated as below.



$x[n]$ $\quad x[\langle n-1\rangle_6] = x[\langle n+5\rangle_6]$ $\quad x[\langle n-4\rangle_6] = x[\langle n+2\rangle_6]$

The middle figure shows $x[n]$ shifted to the right by 1 sample period or, equivalently, shifted to the left by 5 sample periods. The right figure depicts $x[n]$ shifted to the right by 4 sample periods or, equivalently, shifted to the left by 2 sample periods. It should be noted that a circular shift by an integer number n_0 greater than N is equivalent to a circular shift by $\langle n_0 \rangle_N$. To implement the right circular shift and left circular shift, you need to overload `operator >>` and `operator <<`, respectively. The circularly time-reversed sequence of $x[n]$ is defined as $x[\langle -n \rangle_N] = x[\langle N - n \rangle_N]$ implies that $x[\langle -n \rangle_N] = \{\ x[N-n],\ for\ 1 \le n \le N-1\quad x[n],\ for\ n = 0$ which is also a length-N sequence. You need to overload the `unary operator -` to implement the circular time reversal operation. The example output is shown as the following.

Example:

```
#include “Seq.h”
Using namespace std;

int main() {
    Seq seq1("1.dat");
    Seq seq2("2.dat");
    seq1 >> 11;
    seq1.display();
    seq2 << 3;
    seq2.display();
```

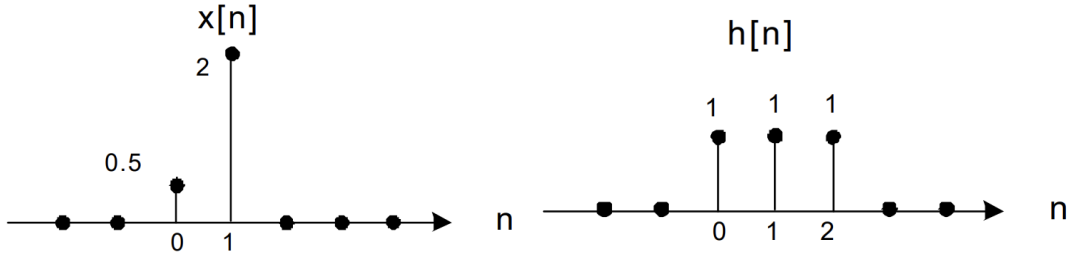
```
    Seq seq3 = Seq(-seq1);
    seq3.display();
    return 0;
}
```

Example output:

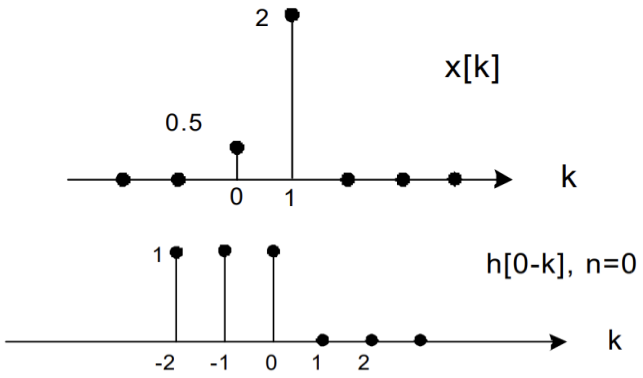
```
1 -1 2 0 -2
0 1 0 1 0 -1
1 -2 0 2 -1
```

C-03 (25%)

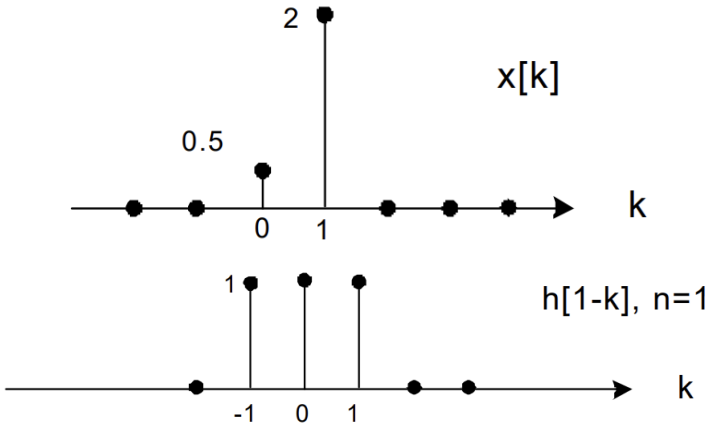
Note that you should complete all the functions in C-02, or you must not do C-03. In C-03, you need to implement the convolution sum operation and an application of discrete-time signal processing, a median filter. Let x[n] and h[n] denote two sequences. The sequence y[n] generated by the **convolution sum** of these two sequences is given by $y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$. To implement the convolution sum operation, you need to overload the `operation *`. It takes two sequences as input whose lengths are N and M respectively, and the output will be a convolution sum of length N+M-1. The detail of convolution sum is explained as the following. Given two sequences x[n] and h[n].



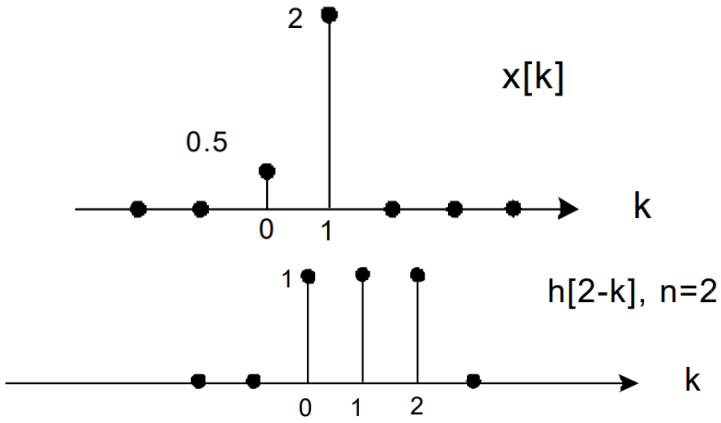
At first, change the variable from n to k, and h[-k] can be derived by h[k] flipping through the y-axis (x=0). By multiplying each sample of two sequences, you can get y[0] directly. Shifting h[-k] to the right by 1 sample at one time, you can finally get the convolution sum sequence. The procedure is shown as below.



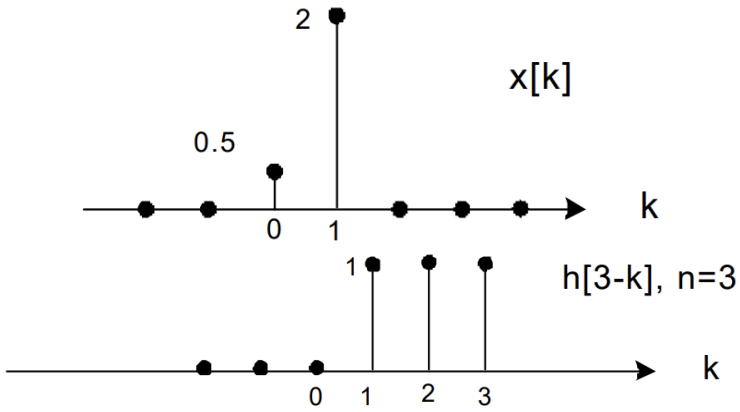
$$y[0] = x[0] \times h[0-k] = x[0]h[0] = 0.5$$



$$y[1] = x[0]h[1] + x[1]h[0] = 0.5 + 2 = 2.5$$



$$y[2] = x[0]h[2] + x[1]h[1] = 0.5 + 2 = 2.5$$



$$y[3] = x[1]h[2] = 2$$

$$y[n] = \{0.5, 2.5, 2.5, 2\}\ whose\ length\ is\ 2 + 3 - 1 = 4$$

To implement a median filter, you have to complete the function `Seq Seq::mid_filter(int M)` which has an **odd** integer parameter M, the median filter size, returns a sequence which is the original sequence after median filtering whose size remains the same. At first, you need to pad $\frac{M-1}{2}$ zeros at the front and at the end of the original sequence. Next on, you have to take M samples as a group, popping out the median of the group, and move the filter to the right by 1 sample until all the samples of the padding sequence have been seen at least once. The details are explained as follows. Suppose the input x[n] = {1, 5, 0, -1, 3}, the median filter size M=3, and the output sequence is y[n]. First, padding the sequence, and you'll get x'[n] = {**0**, 1, 5, 0, -1, 3, **0**}. Each sample of the output sequence can be derived by popping out each group's median. y[0] = median{0, 1, 5} = 1, y[1] = median{1, 5, 0} = 1, y[2] = median{5, 0 ,-1} = 0, y[3] = median{0, -1, 3} = 0, y[4] = median{-1, 3, 0} = 0. At last, you can get y[n] = {1, 1, 0, 0, 0} whose size is the same as the original sequence. The example output is shown as the following.

Example:

```
#include “Seq.h”
Using namespace std;

int main() {
    Seq seq1("1.dat");
    Seq seq2("2.dat");
    seq1 >> 11;
    seq2 << 3;
    seq1.display();
    seq2.display();
    Seq seq3 = seq1 * seq2;
    seq3.display();
    Seq seq4 = seq3.mid_filter(3);
    seq4.display();
    return 0;
}
```

Example output

```
1 -1 2 0 -2
0 1 0 1 0 -1
0 1 -1 3 -1 -1 1 -4 0 2
0 0 1 -1 -1 -1 -1 0 0 0
```

【PROBLEM SET D】(60%)

In this problem, you need to use TA's files to complete a program that simulates the custom mining process of block chain. The command line format shows as below.

$ ./p04 <transaction file path> <miner file path> <simulation time>

Attention

- TA's files are placed in ***/home/share/mid/D/.***
- You need to copy all TA's files to your home directory.
- You can't **add new files** and **rename TA's files**.
- You can **add new functions** in each class TA offers.
- Each miner's transaction array **is filled** during simulation time.
- The order of mining is updated according to the miner's id **in ascending order**.

Mempool is used to keep track of all transactions and coordinate all miners. To complete the functionality of Mempool, you must first parse the transaction file. A transaction record is represented by each line in the transaction file, and its format is as follows:

<id> <fee> <buyer name> <seller name> <coin Amount> <US Dollar>

Second, Mempool has complete control over all miners and has direct access to them. To store miners in Mempool, you must parse the miner file. Each line in the miner file represents a miner, with the following format:

<id> <work type> <computing power>

The id field in the above format is the same as the current line number, and the value of the work type field is 0 (ascending mode) or 1 (descending mode).

Next, the parsing process will be done, and the program will start a simulation of the process of mining the block chain. The main function will utilize an iteration as a timestamp. All miners need to copy two transactions (TRANS_VEC_SIZE) from the Mempool to start. If the work type is set to 0, the miner duplicates the two transactions with the highest fees, and vice versa. After copying transactions, each miner will start to mine the bundle of transactions that has been copied together. According on a miner's computing power, the following formula can be used to determine how long it will take to finish this bundle.

Processing time = 10 – computing power

When the mining process is complete, the miner will notify Mempool in order to receive the reward. Mempool will pay the miner the total fees associated with the completed bundle and then check for other miners. If a miner also mines completed transactions, Mempool will stop the miner's process and adjust the type of work it performs (0->1 or 1->0). Following that, Mempool will delete completed transactions, and any miners who are not mining will duplicate two transactions from Mempool. These actions all occur at the same timestamp.

D-01 (15 points)

Compile all files successfully and run the main files.
Hint:

- add something in all header files
- modify parse functions in functions.cpp

```
$ make clean
$ make
$ make d01
Miner 1's revenue: 0
Miner 2's revenue: 0
Miner 3's revenue: 0
Miner 4's revenue: 0
```

D-02 (25 points)

Complete custom mining process of block chain.

```
$ make clean
$ make
$ make d02
Miner 1's revenue: 3
Miner 2's revenue: 0
Miner 3's revenue: 0
Miner 4's revenue: 78
```

D-03 (20 points)

No memory leak.

```
$ make clean
$ make
$ make d03
…
==###== HEAP SUMMARY:
==###==   in use at exit: 0 bytes in 0 blocks
==###==  total heap usage: $1 allocs, $2 frees, ### bytes allocated
…
```

- If the value of $1 is same as the value of $2, it means that the program has no memory leak.