UEE1303(1048) S20
Midterm Examination

FULL SCORES:

120 %

EXAMINATION TIME:

18：30～21：30，total 180 minutes

INSTRUCTIONS:

This midterm examination is composed of 6 different problem sets. **Your total score will be the summation of the scores from the highest 3 problem sets you've solved.** You are allowed to open any notes or books, but prohibited to browse on the internet to search C/C++ codes as direct answers. Read carefully the statements and requirements of each problem. Once you complete your program for one problem, please raise your hand and TA will come to you and test your program. Please note that points will be given until your program fully fulfills the requirements of each problem. No partial credits will be given.

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

Good luck!

UEE1303(1048) S20: Midterm Examination Demo Sheet

Student ID # : ________________________,Name :__________________________

Full Scores: up to 120 points

→ You may pick arbitrary numbers of problems to solve. **The total score you obtain will be the summation of the scores from the highest 3 problem sets you've solved.**

Problem Set 1 (60%)		**Problem Set 2 (60%)**	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (15%)		Q1 (15%)	
Q2 (25%)		Q2 (20%)	
Q3 (20%)		Q3 (25%)	

Problem Set 3 (60%)		**Problem Set 4 (60%)**	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (15%)		Q1 (25%)	
Q2 (25%)		Q2 (15%)	
Q3 (20%)		Q3 (20%)	

Problem Set 5 (60%)		**Problem Set 6 (60%)**	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (15%)		Q1 (10%)	
Q2 (25%)		Q2 (25%)	
Q3 (20%)		Q3 (25%)	

Total Score: ______________ / 120

【SOURCE CODE AND TESTCASE】

The path of the source code and the open testcase for each problem set:
`<PATH> /tmp/OOP_Midterm/p1 ~ /tmp/OOP_Midterm/p6`
Copy the all the source codes and open testcases to your local directory and start to complete each problem in the corresponding folder.
`<EX> $` **`cp -R /tmp/OOP_Midterm ~/.`**

【PROBLEM SET 1】(60%)

Please complete the definitions of class `Square` in `Square.h` to support the correct execution of the main program (written in `main.cpp`).

You should complete the file `Square.cpp` and add proper prototypes in `Square.h`.

Rotation matrix wiki: https://en.wikipedia.org/wiki/Rotation_matrix

```
//Square.h
#include "Center.h"
#define PI 3.14159265

class Square {

private:
        Center center;
        double len;
        double angle;
        //You can add any private variables if needed
public:
  //Q1 (15%): Basics
    //1-1: Basic class operations
    Square(double, double, double, double);
    Square(const Square &);
    //overload operators << for print every point in square
    //i.e. cout << s1, print information of square s1
    //you should print four points in order.
    //order sequence is: {left, down}, {left, up}, {right, up},
{right, down}
    //if angle is not 0, you should use this order sequence to
calculate coordinate of point and then rotate the square, in the
end, print points in order.

  //Q2 (25%): Overload operators
    //2-1: overload operators + for offset of Square
    //i.e. s1 = s1 + 5, the x,y of square add 5
    //2-1: overload operators + for offset of Square
    //i.e. Center offset = Center(3,2);
```

```
   // s1 += offset, the x of square add 3, the y of square add 2
   //2-2: overload operator * for scale of Square
   //i.e. s1 = s1*2, the new square is twice the area of origin.
   //2-3: overload operator ^ for rotation of Square
   //i.e. s1 = s1^45, rotate square 45 degrees counter-clockwise
from center of square

  //Q3 (20%): Prefix,Postfix,Compare
   //3-1: overload operator ++ (include s1++, ++s1)
   //i.e. s1++ the x,y of square add 1
   //3-2: overload operator -- (include s1--, --s1)
   //i.e. s1--, the x,y of square sub 1
   //3-3: overload operator ==
   //i.e. s1 == s2, compare all content between two square, be
careful of angle, 270 degree == 90 degree
   //3-4: overload operator < for
   //i.e. s1 < s2, check if the area of s2 is bigger than s1
};
```

```
//Center.h

class Center{
   public:
      Center(double x, double y):x(x), y(y){}
      Center(){}
      double x;
      double y;
};
```

Compile: `g++ -o main Square.cpp Center.cpp main.cpp`

Execute: `./main`

Q1 BASICS (10%)

Please complete all constructors and the required member functions in each above class definition with the following requirements. Given the test part Q1 in the main function for this problem:

```
//Q1 tests
     double center_x = 6, center_y = 4, len = 4, angle =0;
     Square s1 = Square(center_x, center_y, len, angle);
     Square s2 = Square(s1);
     Square s3 = Square(s1);
     Square s4 = Square(s1);

     cout << s1;
     cout << s2;
```

Accordingly, the correct output of Q1 part will be

(Ex)

```
Q1 Part:
center:(6,4) len:4 angle:0
point1: 4 2 point2: 4 6 point3: 8 6 point4: 8 2
center:(6,4) len:4 angle:0
point1: 4 2 point2: 4 6 point3: 8 6 point4: 8 2
```

Q2 OVERLOAD OPERATORS (20%) (YOU MUST FINISH Q1 FIRST.)

Given the test part Q2 in the main function for this problem:

```
 //Q2 tests
    s1 = s1 * 2;
    s2 = s2^45;
    s3 = s3 + 5;
    Center offset = Center(3,2);
    s4 += offset;

    cout << s1;
    cout << s2;
    cout << s3;
    cout << s4;
```

Accordingly, the correct output of Q2 part will be

(Ex)

```
Q2 Part:
center:(6,4) len:8 angle:0
point1: 2 0 point2: 2 8 point3: 10 8 point4: 10 0
center:(6,4) len:4 angle:45
point1: 6 1.17157 point2: 3.17157 4 point3: 6 6.82843 point4:
8.82843 4
center:(11,9) len:4 angle:0
point1: 9 7 point2: 9 11 point3: 13 11 point4: 13 7
center:(9,6) len:4 angle:0
point1: 7 4 point2: 7 8 point3: 11 8 point4: 11 4
```

Q3 PREFIX, POSTFIX, COMPARE (30%) (YOU MUST FINISH Q1 FIRST.)

Given the test part Q3 in the main function for this problem:

```
 //Q3 tests
    cout << s1++;
    cout << ++s1;
    cout << s1--;
    cout << --s1;
    cout << (s1 == s2) << endl;
    cout << (s2 < s1) << endl;
```

Accordingly, the correct output of Q3 part will be

(Ex)

```
Q3 Part:
center:(6,4) len:8 angle:0
point1: 2 0 point2: 2 8 point3: 10 8 point4: 10 0
center:(8,6) len:8 angle:0
point1: 4 2 point2: 4 10 point3: 12 10 point4: 12 2
center:(8,6) len:8 angle:0
point1: 4 2 point2: 4 10 point3: 12 10 point4: 12 2
center:(6,4) len:8 angle:0
point1: 2 0 point2: 2 8 point3: 10 8 point4: 10 0
0
1
```

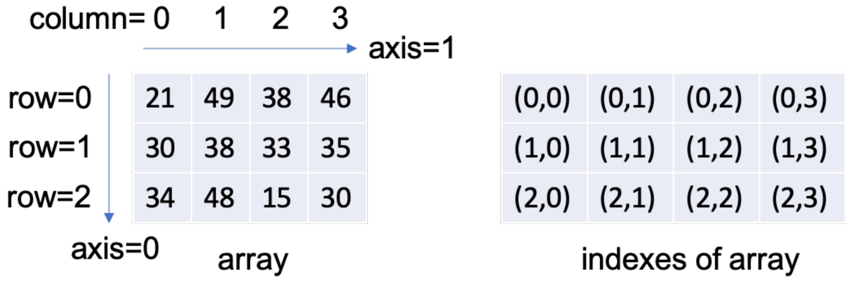
==

【PROBLEM SET 2】(60%)

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for array processing.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

In this problem set, you should use **two-dimensional arrays** in C++ to implement some functions of array in NumPy package. Please finish the definition of class `numpy` to make sure the main program match answers in each problem. You shouldn't handle any errors about test data if you actually finish these functions correctly. There are more details of descriptions below.



21	49	38	46
30	38	33	35
34	48	15	30

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

The dimensions and index values of the array are defined in this figure.

You can only **modify** `numpy.h` and `numpy.cpp`, but **NOT** `main.cpp`.

Q1 BASICS (15%)

Please finish `/* Q1 Basics */` in `numpy.h` and `numpy.cpp` .

Functions:
At first, you should make programs be able to read **a binary file**. The first two numbers in this file represent rows and columns of arrays.
For example, you can set your size of array with these numbers and create a 3*4 array. “3” means rows and “4” means columns. Other numbers in this file are values in this array you create as shown in the figure above. Therefore, total fourteen numbers should be read in your program. All numbers are **integer types and positive**. Then, please make these functions work correctly.

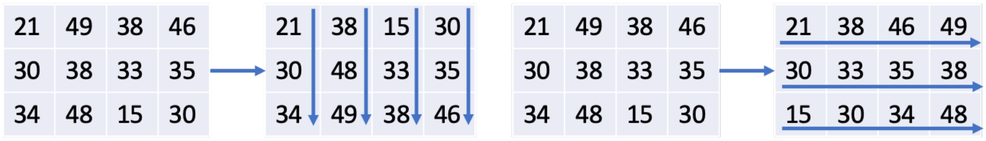
`max()`: Return the maximum value in an array.
`argmax()`: Return the index of maximum value **first** found in an array. If there are three same maximum values in indexes of (0, 6), (1, 2) and (1, 5) respectively. `argmax()` will return 0 and 6.

(Ex) Compile: `g++ -o main_1 main_1.cpp numpy.cpp`

```
> ./main_1 array_1.dat ↵
max = 49
argmax = (0, 1)
>
```

Q2 MODERATE (20%) (YOU MUST FINISH **Q1** FIRST.)
Please finish **`/* Q2 Moderate */`** in `numpy.h` and `numpy.cpp`.
In this part, you should implement these functions below.

Functions:
overloading operator []: Return the value corresponding to the index.
overloading operator <<: Print out every value in an entire array. Please leave a space between next value in the same row and there is **no space in head and end of row**.
`sort(int axis)`: Return the values in an array from small to large in the specified axis. When axis is equal to 0 or1, there will be different results like the figure below.



21	49	38	46
30	38	33	35
34	48	15	30

21	38	15	30
30	48	33	35
34	49	38	46

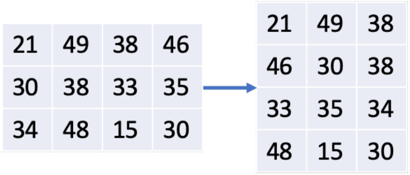
sort(axis=0)

21	49	38	46
30	38	33	35
34	48	15	30

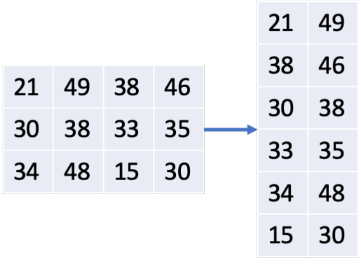
21	38	46	49
30	33	35	38
15	30	34	48

sort(axis=1)

`reshape(int n, -1)`: Return a new array with all the original values but in a different shape. n means the column value and -1 means “don’t care”. The example is like below.



reshape(3, -1)



reshape(2, -1)

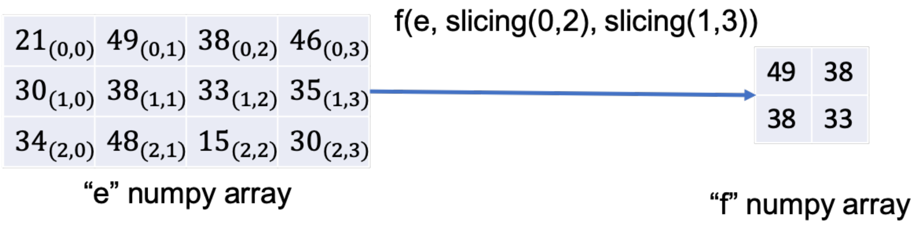
(Ex) Compile: `g++ -o main_2 main_2.cpp numpy.cpp`

```
> ./main_2 array_1.dat ↵
38
21 49 38 46
30 38 33 35
34 48 15 30

21 38 15 30
30 48 33 35
34 49 38 46

15 21 30 38
30 33 35 48
34 38 46 49

15 21 30
38 30 33
35 48 34
38 46 49

>
```

Q3 ADVANCED (25%) (YOU MUST FINISH **Q2** FIRST.)

Please finish **`/* Q3 Advanced */`** in `numpy.h` and `numpy.cpp`.
In this part, you should implement these functions below.

Functions:
overloading operator *: Return a new array after two arrays are multiplied.
`concatenate(int axis)` : Return a new array by combining two arrays together according to specified axis.
`slicing(int n, int m)` : Return **two integers** in an array. This function is used for the next constructor.
`numpy::numpy(const numpy & a, int* r, int* c)` :Construct a numpy array with slicing functions like the example below. You will get a new array with `slicing(0, 2)` ranging from 0 to 2 in row (not included itself) and `slicing(1, 3)` ranging from 1 to 3 (not included itself) in column.



f(e, slicing(0,2), slicing(1,3))

$21_{(0,0)}$	$49_{(0,1)}$	$38_{(0,2)}$	$46_{(0,3)}$
$30_{(1,0)}$	$38_{(1,1)}$	$33_{(1,2)}$	$35_{(1,3)}$
$34_{(2,0)}$	$48_{(2,1)}$	$15_{(2,2)}$	$30_{(2,3)}$

"e" numpy array

49	38
38	33

"f" numpy array

(Ex) Compile: `g++ -o main_3 main_3.cpp numpy.cpp`

```
> ./main_3 array_1.dat ↵
3517 4133 4025
4753 5508 5531
5426 6316 6239

15 21 30
38 30 33
35 48 34
38 46 49
3517 4133 4025
4753 5508 5531
5426 6316 6239

15 21 30 38 3517 4133 4025
30 33 35 48 4753 5508 5531
34 38 46 49 5426 6316 6239

21 30
33 35

>
```

==

【PROBLEM SET 3】(60%)

Please complete the definitions of class `Polynomial` in `Polynomial.h` to support the correct execution of the main program (written in `main.cpp`).

You should complete `Polynomial.cpp` and add proper prototypes in `Polynomial.h`.

```
//Polynomial.h

class Polynomial {
private:
  int **coef;
  int length;

public:
  // Q1 show function + Constructors
  void show() const;
  Polynomial( int **s, int len);
  Polynomial(const Polynomial &s);
  ~Polynomial()
  // Q2 overload operator +,-(Both subtraction and Unary minus),*
  // Q3 overload operator /,%
};
```

Compile: `g++ -o main Polynomial.cpp main.cpp`

Execute: `./main`

<u>Q1 CONSTRUCTORS AND MEMBER FUNCTIONS</u> (15%)

Please complete all constructors and the required member functions.

Detail: Suppose `int**a` is a square matrix `a[i][j]`.
`i` is the degree of x, and `j` is the degree of y.
`Length` is the matrix width.

Given the test part Q1 in the main function for this problem:

```
   /*Q1 test
   lenA = 3, a[0][0]=3, a[0][1]=0, a[0][2]=-2, a[1][0]=1,
   a[1][1]=2, a[1][2]=-1, a[2][0]=2, a[2][1]=1, a[2][2]=2  */
   Polynomial A(a, lenA);
   A.show();
```

Accordingly, the correct output of Q1 part will be

```
(Ex)
 2x^2y^2+x^2y+2x^2-xy^2+2xy+x-2y^2+3
```

The displayed priority is to consider the higher degree of x, then y, and finally constant.

Q2 OVERLOAD OPERATORS (25%) (YOU MUST FINISH Q1 FIRST.)

Given the test part Q2 in the main function for this problem:

```
   /*Q2 test
   lenB = 2, b[0][0]=0, b[0][1]=0, b[1][0]=0, b[1][1]=-2 */
   Polynomial B(b, lenB);
   (A+B).show();
   (A-B).show();
   (A*B).show();
   (-B).show()
```

Accordingly, the correct output of Q2 part will be

```
(Ex)
  2x^2y^2+x^2y+2x^2-xy^2+x-2y^2+3
  2x^2y^2+x^2y+2x^2-xy^2+4xy+x-2y^2+3
  -4x^3y^3-2x^3y^2-4x^3y+2x^2y^3-4x^2y^2-2x^2y+4xy^3-6xy
  2xy
```

Q3 OVERLOAD OPERATORS (20%) (YOU MUST FINISH Q1 FIRST.)

In this part, to simply the problem divisor will be single variable like x^2+1, or y^3-y+1. And the procedure can all be done by integers.

Given the test part Q3 in the main function for this problem:

```
   /*Q3 test
   lenC  =  2,  c[0][0]=-2,  c[0][1]=1,  c[1][0]=0,  c[1][1]=0  */
   Polynomial C(c, lenC);
   (A/C).show();
   (A%C).show();
```

Accordingly, the correct output of Q3 part will be

```
(Ex)
  2x^2y+5x^2-xy-2y-4
  12x^2+x-5
```

==

【PROBLEM SET 4】(60%)

You are asked to implement an on-line Boolean calculator to derive a factored form of a clean Boolean formula from the given set of inputs (For example, given $\{a, b, c, d, e\}$ as the input set, an output, f, can be expressed into

$$f = (\overbrace{\left(\underbrace{\overbrace{\underbrace{(a' + b)}_{g}\, cd}^{h} + e}_{w}\right)\underbrace{(a + b')}_{x}}^{y}) + e'$$

) by a list of clauses in a file. Each clause is called a Clause term (**Cluz**) and associated with only one Boolean operator either in AND($*$) or OR($+$). For example, you will read the following clauses for f where $\{g, h, w, x, y\}$ are **Cluz**s:

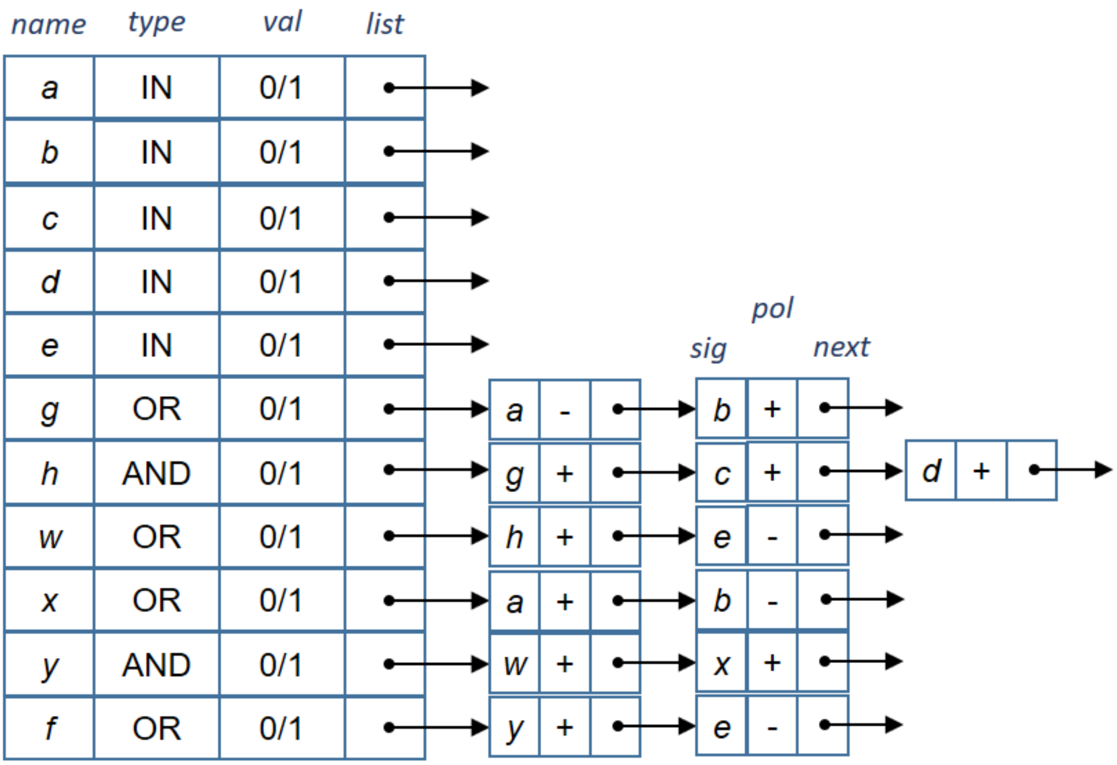
$$\begin{gathered} g = \mathrm{a}' + \mathrm{b} \\ h = g * c * d \\ w = h + e \\ x = a + b' \\ y = w * x \\ f = y + e'. \end{gathered}$$

For ease of your processing, each testcase file will contain the basic information at the beginning. All terms/literals and operators separated by a space. For example,

```
5 6             // 5 inputs + 6 clauses
a b c d e       // input set
g = a’ + b      // literals/operators separated by space
h = g * c * d
w = h + e’
x = a + b’
y = w * x
f = y + e’
```

Note that the final clause is always used as the output formula (e.g. f in this case) that you need to derive. Moreover, these **Cluz**s are properly arranged so that every input of one **Cluz** would appear earlier either as an input or a **Cluz** mentioned above.

Meanwhile, you will be also asked to implement a particular data structure for each **Cluz**, a list of cube terms (**Cube**) to record the polarity of each variable for the inputs, **Cluz**s and output. Take input x as example, two types of the polarities, either in a positive literal (denoted by x only with $+$) or a negative literal (denoted by x' with a $'$ as $-$) can be used in the file. For better understanding the entire data structure, we use the above example again for illustration.



name	type	val	list
a	IN	0/1	
b	IN	0/1	
c	IN	0/1	
d	IN	0/1	
e	IN	0/1	
g	OR	0/1	
h	AND	0/1	
w	OR	0/1	
x	OR	0/1	
y	AND	0/1	
f	OR	0/1	

Since the input in $\{a, b, c, d, e\}$ involves no operation, the list field points to NULL. As to the **Cluz** objects, their ***list*** points to a list of **Cube** terms for inputs involved in their respective Boolean operations. The ***val*** field in each node of the **Cluz** list is used to store the Boolean value during evaluation.

Please read a testcase which contains all Boolean clauses, provide required operations and save the factored form to the target file. The objective of this problem is to remove all clause terms (**Cluz**s) and restore a *clean* Boolean formula only consisting of the inputs for the output. You need to complete the definitions of the clause class **Cluz** and the cube term class **Cube** in **BoolForm.h** to support the correct execution of the main program (written in **main.cpp**). Each object of **Cluz** contains at least 4 fields (***name***, ***type***, ***val*** and ***list***). Field name denotes the name of the clause in ***char***. Field ***type*** can be assigned with one of two simple Boolean operators (either **AND**($*$) or **OR**($+$)) or the input (**IN**). Field ***val*** can only be either 0 or 1. Field ***list*** is a pointer that links the input signals as a list of **Cube** terms in one **Cluz**. Each object of **Cube** contains three fields, ***sig***, ***pol*** and ***next***. Field ***sig*** denotes the name of the signal in ***char***. Field ***pol*** indicates the polarity of the current **Cube** term whereas Field ***next*** links to the next **Cube** term in the current **Cluz**. In principle, you only should complete the functions in **BoolForm.cpp** but feel free to add or modify prototypes in **BoolForm.h** if necessary.

```
 //BoolForm.h
 enum bop {IN, AND, OR}

 class Cube {
    char sig;
    bool pol;
    Cube *next;
 ...
 }

 class Cluz {
    char name;
    bop type;
    bool val;
    Cube *list
 ...
};
```

Q1 BASIC (25%)

Please complete all constructors and the required member functions in each above class definition with the following requirements. More specifically, Q1 must run under the command-line mode for operations. You need to develop three commands, **`read`** for reading the input file, **`list`** for displaying all clauses of intermediate terms and **`exit`** for exiting this system. Accordingly, the correct message on screen will look like

```
>./p4 ↵
read 4-1.in ↵
done opening 4-1.in to read 5 inputs and 6 clauses.
list ↵
OR g a’ b
AND h g c d
OR w h e
OR x a b’
AND y w x
OR f y e’
exit ↵
>
```

Q2 MODERATE (15%)

Q2 mainly asks you to implement the command-mode evaluation function (**`eval`**) on the Boolean formula for the given input assignment.

```
>./p4 ↵
...(reading the input testcase e.g. 4-1.in)...
eval 0100 ↵
[ERROR] input 5 values for 5 inputs
eval 10110 ↵
f(10110)=1
eval 01001 ↵
f(01001)=0
...
```

Q3 ADVANCED (20%)

Q3 mainly asks you to out the derivation of a clean Boolean formula as the factored form in terms of the input set to the target file.

```
>./p4 ↵
...(reading the input testcase e.g. 4-1.in)...
write 4-1.ot ↵
The reduced form of f written in 4-1.ot
exit ↵
>
```

As a result, the file `4-1.ot` will look like:

```
> cat 4-1.ot ↵
f=(((a'+b)*c*d+e)(a+b'))+e'
>
```

==

【PROBLEM SET 5】(60%)

In this problem, you are going to implement a Darray class. Darray is a dynamic array class, which allocates memory on demand. You have to handle two integers "capacity" and "size". Capacity is the total volume of memory you allocated from the system and size is the actual elements saved in the array. If your array is out of memory, you have to allocate a new segment of memory which is two times as large as the original capacity. For example, if a Darray has capacity of 10 and already filled with 10 elements, when you push_back one more element, you have to allocate a new segment of memory with capacity of 20 and move the elements to the new array. If original capacity is 0, then the new capacity will be 1.

The header file `Darray.h` and some example are provided. There are some blanks

left in the header file, you should fill in the correct function declaration. You can add any private variables or functions if needed.
Note that there will be hidden cases for this problem.

Q1 STATIC ARRAY (15%)

In this part, you need to implement the basic functions for Darry in static version. Including constructor, destructor, [] operator,= operator, size, capacity, empty and binary file read/write operations. You can find information of each function in the header file.
Note that:

1. Index operator [] should support assignment (ex: a[0]=87) and access(ex: int x=a[0]). Also, you should handle it for const objects.
2. Spec. for binary file: The first integer will be the size of the Darray. And the rest will be the content of Darray. You can find an example file *ex.dat* with 10 elements from 1 to 10.
3. Remember to comment part2 and part3 in the header file before you compile.

Example: main_part1.cpp

```
#include "Darray.h"
#include <iostream>
void printDarray(const Darray &a){
    for(int i=0;i<a.size();i++)
        std::cout<<a[i]<<" ";   //[]operator access
    std::cout<<"\n";
}

int main(){
    Darray a(5,10);     //Constructor with initial size
and value
    printDarray(a);
    Darray b(5);       //Constructor with initial size
    printDarray(b);
    int x[3]={100,50,30};
    Darray c(x,3);     //Constructor with int array
    printDarray(c);
    Darray d;          //Default Constructor
    d.read("ex.dat");   //read binary file
    printDarray(d);
    d[0]=d[1]=5;       //[]operator and assignment
    printDarray(d);
    const Darray e(5,5);//Const object
    printDarray(e);
    Darray f(e);       //Copy Constructor
    f[0]=10;
    printDarray(f);
}
```

Example output for main_part1.cpp

```
10 10 10 10 10
0 0 0 0 0
100 50 30
1 2 3 4 5 6 7 8 9 10
5 5 3 4 5 6 7 8 9 10
5 5 5 5 5
10 5 5 5 5
```

Q2 DYNAMIC ARRAY (25%) (YOU MUST FINISH Q1 FIRST.)

In this part, you need to implement the functions related to dynamic array operations. Including resize, reserve, push_back, pop_back, operator+, operator+=. You can find information of each function in the header file.

Note that:

1. You have to handle the memory for dynamic array by yourself.
2. For both operators + and +=, you should provide integer and Darray versions.
3. You have to fill in the function declaration for operator overloading by yourself.
4. Remember to comment part3 in the header file before you compile.

Example: main_part2.cpp

```
#include "Darray.h"
#include <iostream>
using namespace std;
void printDarray(const Darray &a){
    for(int i=0;i<a.size();i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    cout<<"Size: "<<a.size()
         <<" Capacity: "<<a.capacity()<<endl;
}

int main(){
    Darray a;
    printDarray(a);
    for(int i=0;i<10;i++){
        a.push_back(i);
        printDarray(a);
    }
    a.pop_back();
    a.pop_back();
    printDarray(a);
    a.resize(20);
    printDarray(a);
    a.reserve(32);
    printDarray(a);
    Darray b(3,5);
    Darray c(5,6);
    Darray d=b+c;
    printDarray(d);
    d=d+0;
    printDarray(d);
    d+=10;
    printDarray(d);
    d+=c;
    printDarray(d);
}
```

Example output for main_part2.cpp

```
Size: 0 Capacity: 0
0
Size: 1 Capacity: 1
0 1
Size: 2 Capacity: 2
0 1 2
Size: 3 Capacity: 4
0 1 2 3
Size: 4 Capacity: 4
0 1 2 3 4
Size: 5 Capacity: 8
0 1 2 3 4 5
Size: 6 Capacity: 8
```

```
0 1 2 3 4 5 6
Size: 7 Capacity: 8
0 1 2 3 4 5 6 7
Size: 8 Capacity: 8
0 1 2 3 4 5 6 7 8
Size: 9 Capacity: 16
0 1 2 3 4 5 6 7 8 9
Size: 10 Capacity: 16
0 1 2 3 4 5 6 7
Size: 8 Capacity: 16
0 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0
Size: 20 Capacity: 20
0 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0 0 0 0 0
Size: 20 Capacity: 32
5 5 5 6 6 6 6 6
Size: 8 Capacity: 8
5 5 5 6 6 6 6 6 0
Size: 9 Capacity: 9
5 5 5 6 6 6 6 6 0 10
Size: 10 Capacity: 18
5 5 5 6 6 6 6 6 0 10 6 6 6 6 6
Size: 15 Capacity: 18
```

Q3 Utility Functions (20%) (You Must Finish Q2 First.)

In this part, you need to implement functions including insert, erase, clear, swap. You can find information of each function in the header file.

Example: main_part3.cpp

```
#include "Darray.h"
#include <iostream>
using namespace std;
void printDarray(const Darray &a){
   for(int i=0;i<a.size();i++)
      cout<<a[i]<<" ";
   cout<<"\n";
   cout<<"Size: "<<a.size()
      <<" Capacity: "<<a.capacity()<<endl;
}
```

```
int main(){
    Darray a;
    a.read("ex.dat");
    printDarray(a);
    a.insert(3,66);
    printDarray(a);
    a.insert(10,87);
    printDarray(a);
    a.erase(10);
    printDarray(a);
    a.erase(3,6);
    printDarray(a);
    Darray b;
    b.read("ex.dat");
    a.swap(b);
    printDarray(a);
    printDarray(b);
    a.clear();
    printDarray(a);
}
```

Example output for main_part3.cpp

```
1 2 3 4 5 6 7 8 9 10
Size: 10 Capacity: 10
1 2 3 66 4 5 6 7 8 9 10
Size: 11 Capacity: 20
1 2 3 66 4 5 6 7 8 9 87 10
Size: 12 Capacity: 20
1 2 3 66 4 5 6 7 8 9 10
Size: 11 Capacity: 20
1 2 3 7 8 9 10
Size: 7 Capacity: 20
1 2 3 4 5 6 7 8 9 10
Size: 10 Capacity: 10
1 2 3 7 8 9 10
Size: 7 Capacity: 20

Size: 0 Capacity: 10
```

【PROBLEM SET 6】(60%)

In this problem, you are going to implement three classes (Base, Material, and Cocktail). Base is the class which describes the basic information of materials that used to make the cocktail. In class Material, it records the amount and material that used in the cocktail. The Cocktail class contains the name of cocktail and the materials of the cocktail.
The source code (Cocktail.h and main.cpp) and input files (base.in and cocktail.in) will be given. Some details of the classes will be written in the comments. Please add proper codes to Cocktail.h and complete Cocktail.cpp. But Do Not modified main.cpp and input files.

```
<Compile> $ g++ main.cpp Cocktail.cpp -o p6
<Usage> $./p6 <base file> <cocktail file> <question choice>
(Ex.) $ ./p6 base.in cocktail.in 1
```

Header file: Cocktail.h

```
#ifndef _COCKTAIL_H_
#define _COCKTAIL_H_

#include<string>
using namespace std;
class Base{
  private:
    string name;    // name of material
    double alcohol; // alcohol concentration
    double sweet;   // sweet flavor (0~10)
    double sour;    // sour flavor (0~10)
    double bitter;  // bitter flavor (0~10)
  public:
  //Q1 Basic
    //Q1-1
    Base();
    //Q1-2
    Base(const Base&);
    //Q1-3
    //Use sigle line string to initialize the Base object
    Base(string line);
    //Q1-4
    //precision setting -> only 1 digit
    //width setting: name -> 12, others -> 5
    friend ostream& operator<<(ostream& os, const Base& b);
};

class Material{
  private:
    Base base;
    int amount; //amount of the base material in milliliter (ml)
  public:
  //Q2 Moderate
    //Q2-1
    Material();
    //Q2-2
    //Use single line string to initialize the Material object
    //The Base pointer array (b) and the size of array (n) are
given
    Material(string line, Base* b, int n);
    //Q2-3
    friend ostream& operator<<(ostream& os, const Material& m);
  //Q3 Advanced
    //Q3-1
    //Add another material into the existed one
    //The alcohol, sweet, sour, bitter should be calculated
    //linearly with the amount of the material
    void operator+=(const Material& m);
};

class Cocktail{
  private:
    string name;  //Name of the cocktail
    int num;      //number of materials
    Material* m;  //materials
  public:
  //Q1 Basic
    //Q1-7
    Cocktail();
    //Q1-8
    //Use Material pointer array (M), number of materials (N),
    //and name of cocktail (Name) to initialize the Cocktail object
    Cocktail(string NAME, int N, Material* M);
  //Q2 Moderate
    //Q2-4
    //print out the recipe of the cocktail
    friend ostream& operator<<(ostream& os, const Cocktail& c);
  //Q3 Advanced
    //Q3-2
    //Mix all the materials together
    //Hint: It would call the += operator of class Material
    Material operator++(int a);
};
#endif
```

Input file: `base.in`

```
//Name Alcohol Sweet Sour Bitter
Num 3
Wiskey       56  0 0 0
LemonJuice    0   0 5 1
Syrup         0   5 0 0
```

Input file: `cocktail.in`

```
Num 1

Cocktail Wiskey_Sour 3
Wiskey     60
LemonJuice  15
Syrup      15
End
```

Main Program: `main.cpp`

```
#include"Cocktail.h"
#include<iostream>
#include<fstream>
#include<cstdlib>
```

```
#include<sstream>
using namespace std;

int main(int argc, char** argv)
{
  ifstream fin;
  stringstream ss;
  string line, tmp, cockName;
  size_t found;
  bool flag;
  int question_ch;
  int num_base, num_cock, num_mat, amount, index_b, index_m,
index_c;
  Base* base;
  Base* b;
  Material* material;
  Material* m;
  Cocktail* cocktail;
  Cocktail* c;

  fin.open(argv[1]);
  index_b = 0;
  while(getline(fin, line))
  {
    found = line.find("//");
    if(found != string::npos)
      continue;
    found = line.find("Num");
    if(found != string::npos)
    {
      tmp = line.substr(found+4,line.length());
      num_base = atoi(tmp.c_str());
      base = new Base [num_base];
      continue;
    }
    b = new Base(line);
    base[index_b] = *b;
    index_b++;
  }
  fin.close();

  index_m = index_c = 0;
  fin.open(argv[2]);
  while(getline(fin, line))
  {
    if(line.find("Num") != string::npos)
    {
      found = line.find("Num") + 4;
      tmp = line.substr(found,line.length());
      num_cock = atoi(tmp.c_str());
      cocktail = new Cocktail [num_cock];
      index_c = 0;
      continue;
    }
    if(line.find("End") != string::npos)
    {
      c = new Cocktail(cockName, num_mat, material);
      cocktail[index_c] = *c;
      index_c++;
      flag = false;
    }
    if(line.find("Cocktail") != string::npos)
    {
      ss.clear();
      ss << line;
      ss >> tmp;
      ss >> cockName;
      ss >> num_mat;
      material = new Material [num_mat];
      index_m = 0;
      flag = true;
      continue;
    }
    if(flag == true)
    {
      m = new Material(line, base, num_base);
      material[index_m] = *m;
      index_m++;
    }
  }
  fin.close();

  question_ch = atoi(argv[3]);

  //Basic
  if(question_ch == 1)
    for(int i=0; i<num_base; i++)
      cout << base[i] << endl;
  //Moderate
  if(question_ch == 2)
    for(int i=0; i<num_cock; i++)
      cout << cocktail[i] << endl;
  //Advanced
  if(question_ch == 3)
    for(int i=0; i<num_cock; i++)
      cout << cocktail[i]++ << endl;
  return 0;
}
```

Q1 BASICS (10%)

In this part, you need to implement the basic functions in `Cocktail.cpp` for each class that declared in the `Cocktail.h` to make the main program execute successfully.

Example: Q1 of main.cpp

```
//Basic
  for(int i=0; i<num_base; i++)
    cout << base[i] << endl;
```

Example output for Q1 of main.cpp

Please set precision with only one digit to align the output results.

For the names of materials, please set width (12).

For the other flavors, please set width (5).

```
      Wiskey 56.0  0.0  0.0  0.0
  LemonJuice  0.0  0.0  5.0  1.0
```

```
      Syrup  0.0  5.0  0.0  0.0
```

Q2 MODERATE (25%) (YOU MUST FINISH Q1 FIRST.)

In this part, you need to complete the operator overloading function << of all the classes to print out the correct recipe of all the cocktails.

Example: Q2 of main.cpp

```
//Moderate
  for(int i=0; i<num_cock; i++)
    cout << cocktail[i] << endl;
```

Example output for Q2 of main.cpp

The precision setting is the same as Q1.

```
Wiskey_Sour (# of Materials: 3)
     Wiskey 56.0  0.0  0.0  0.0 (60 ml)
  LemonJuice  0.0  0.0  5.0  1.0 (15 ml)
      Syrup  0.0  5.0  0.0  0.0 (15 ml)
```

Q3 ADVANCED (25%) (YOU MUST FINISH Q2 FIRST.)

In this part, you need to implement functions including operator++ of Cocktail class and operator += of Material class. The operator++ mix all the materials in the cocktail, it will return a Material object. And operator += of Material is used to mix two materials together. The alcohol concentration and value of flavors will be calculated linearly with the amount of the material.

Example: Q3 of main.cpp

```
//Advanced
  for(int i=0; i<num_cock; i++)
    cout << cocktail[i]++ << endl;
```

Example output for Q3 of main.cpp

```
Wiskey_Sour 37.3  0.8  0.8  0.2 (90 ml)
```