Assignment 2 (10%)

Due Date: 30 Nov 2025, 23:59 PM

1. This assignment contains **3** questions. You are required to finish a complete C++ program (.cpp only) for each question and submit via Canvas before the deadline.

2. You are strongly encouraged to carefully check your program on any computer in AC1-201 before the submission. If your submitted solution leads to a compilation error, a mark of ZERO will be given to that question.

3. The marking of each question is based on the percentage of the total test cases that your solution can pass. Try your best to thoroughly check your program for different test case scenarios.

4. No late submission will be accepted.

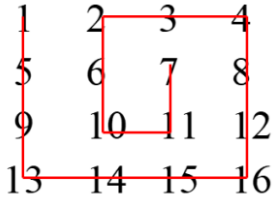
5. Plagiarism check will be performed.

6. You only need to use the material from Lectures 1 to 9. It is **NOT** necessary to include any other library, except <iostream> and <string>.

7. You need to check your solution and submit it via Canvas.

Q1: [25%]

Write a program that takes a **square** matrix (i.e., the number of rows equals to the number of columns) as input and prints all its elements in a **spiral** form, following an **anticlockwise** direction. An example



is given in the figure, where the **size** of this matrix (i.e., the number of rows or columns) is 4. The printed output of all its elements in an anticlockwise spiral form is:

1 5 9 13 14 15 16 12 8 4 3 2 6 10 11 7

Notes:

1. The input matrix size is an integer-type value
2. We assume the input matrix size is correct, i.e., no need to check its correctness
3. We assume that the maximum size of the input matrix is 10

Example 1:

```
Please input the size of the matrix:
1
Please input the matrix row by row:
6
The spiral form of the matrix is:
6
```

Example 2:

```
Please input the size of the matrix:
4
Please input the matrix row by row:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
The spiral form of the matrix is:
1 5 9 13 14 15 16 12 8 4 3 2 6 10 11 7
```

Example 3:

```
Please input the size of the matrix:
5
Please input the matrix row by row:
1 2 3 4 5
6 7 8 9 1
2 3 4 5 6
7 8 9 1 2
3 4 5 6 7
The spiral form of the matrix is:
1 6 2 7 3 4 5 6 7 2 6 1 5 4 3 2 7 3 8 9 1 5 9 8 4
```

Q2: [35%]

Implement a management system for hotel rooms using two classes, Room and Hotel, with the functionalities described below. Use the <string> library to facilitate your implementation.

Class 1: Room

Member Variables:

- name (string): Room identifier (e.g., "room1").
- floor (string): Floor number (e.g., "Floor2").
- price (int): Price per night.
- status (int): 1 (available) or 0 (booked).

Member Functions:

1. void set(string n, string f, int p, int s): Initialize name, floor, price, and status.
2. void book(string name):
    - If the room matches the input name and this room is available (status == 1), set status to 0 and print "Succeed!".
    - If already booked, print "Sorry, the room is not available. Try again.".
3. void checkout(string name):
    - If the room matches the input name and is booked (status == 0), set status to 1 and print "Thanks for using our services!".
    - If not booked, print "This room is not booked yet!".
4. void print(): Output the room's name, floor, price, and status in one line, separated by a space.
5. bool operator>(const Room &r): Overload > to compare rooms:
    - Return true if this room's price is higher than r's.
    - If prices are equal, return true if this room's name is lexicographically greater than r's, e.g., "room2" is greater than "room1". (hint: use compare() from the <string> library.)

Class 2: Hotel

Member Variables:

- rPtr (pointer to Room): Dynamically allocated array of rooms.
- num (int): Number of rooms in the array.

Private Member Function:

1. void sortRooms(): Sort rPtr in ascending order and implement the sorting by yourself. Use the operator> overload for comparisons.

Public Member Functions:

1. Hotel(int n): Constructor to allocate rPtr with n rooms and set num = n.
2. ~Hotel(): Destructor to deallocate rPtr.
3. void setRoom(int idx, string n, string f, int p, int s): Initialize the room at index idx using Room::set().
4. void bookRoom(string rname): Attempt to book a room with name rname.
5. void checkoutRoom(string rname): Attempt to check out a room with name rname.
6. void listRooms(): Print all rooms after sorting them with sortRooms().

Notes:

- Initial codes and example are provided below.
- We assume that all inputs are correct. No need to check the correctness of the input.

Code Framework

```
// TODOs
int main() {
    int n;
    cout << "Input the number of rooms:" << endl;
// TODOs
    string cmd, inf;
    do {
        cout << "Input the instruction:" << endl;
        cin >> cmd;
        if (cmd == "Book") {
            cin >> inf;
            // TODO: book using inf
        } else if (cmd == "Return") {
            cin >> inf;
            // TODO: checkout using inf
        } else if (cmd == "List") {
            // TODO:
        } else if (cmd == "Leave") {
            // TODO:
        }
        cout << endl;
    } while(true);
    return 0;
}
```

Example 1:

```
Input the number of rooms:
9
Input the information of each room (room name, floor, price, status):
room1 Floor1 320 0
room2 Floor1 200 0
room3 Floor2 200 1
room4 Floor3 200 1
room5 Floor1 320 0
room6 Floor3 400 1
room7 Floor3 400 0
room9 Floor2 150 1
room8 Floor2 150 0

All rooms in the hotel:
room8 Floor2 150 0
room9 Floor2 150 1
room2 Floor1 200 0
room3 Floor2 200 1
room4 Floor3 200 1
room1 Floor1 320 0
room5 Floor1 320 0
room6 Floor3 400 1
room7 Floor3 400 0

Input the instruction:
Book room1
Sorry, the room is not available. Try again.
```

```
Input the instruction:
Book room3
Succeed!

Input the instruction:
Return room3
Thanks for using our services!

Input the instruction:
Return room3
This room is not booked yet!

Input the instruction:
List
All rooms in the hotel:
room8 Floor2 150 0
room9 Floor2 150 1
room2 Floor1 200 0
room3 Floor2 200 1
room4 Floor3 200 1
room1 Floor1 320 0
room5 Floor1 320 0
room6 Floor3 400 1
room7 Floor3 400 0

Input the instruction:
Leave
See you again!
```

Q3: [40%]

Write a program that simulates a game of drawing **1~3** balls from a bag. The program should allow the user to enter the number of draws they want to make, then generate and display all possible combinations of drawn numbers, along with the number of occurrences for each sum of values. There are 6 balls in total and each one has a unique ID: 1, 2, 3, 4, 5, and 6. After each draw, the ball is returned to the bag (with replacement).

The code framework is given. What you need to do is following:

Requirements:

1. Implement the `findSum1, findSum2, findSum3` methods in the Combination class to correctly calculate and store combinations of drawn numbers based on the number of draws.
2. Implement the `sortBySum` member function to sort combinations in **descending order** by total sum. (If the total sums are the same, further sort them by their occurrence counts in **descending order**.)
3. Implement the `sortByOcc` member function to sort combinations in **descending order** by occurrence count. (If occurrence counts are the same, further sort them by their total sums in **descending order**.)
4. You may add helper member functions to the Combination class as needed.
5. Test the program with **1~3** draws and ensure correct calculations.

Additional challenging question:
Implement a member function `findSum4to12` to handle **4~12** draws. Some test cases will need to use this function.

Hint: 1) cNum represents the total number of distinct sum values when drawing balls.
2) You may need to use recursion for 4~12 draws. The relationship between N draws and N+1 draws is simply add a ball.

Note: No validity check is required for the inputs.

Code Framework

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

class Combination {
   private:
       int dNum;
       int cNum;
       int **pPtr;
   public:
       Combination(int i = 1) {
/************** to be finished **************/
   }
       ~Combination() {
/************** to be finished **************/
/***********Remember to release memory*******/

       cout << "Memory is released" << endl;
       }
```

```
        void findSum1();
        void findSum2();
        void findSum3();

/************** to be finished **************/

        void sortBySum();
        void sortByOcc();
};


void Combination::findSum1() {

}

void Combination::findSum2() {

}
void Combination::findSum3() {

}


void Combination::sortBySum() {

}

void Combination::sortByOcc() {

}

void display(Combination &com) {

   int n;
   do {
       cout << "~~~~~~~~~~~~~~~~~" << endl;
       cout << "0 exit" << endl;
       cout << "1 sort by sum" << endl;
       cout << "2 sort by occurrence" << endl;
       cout << "~~~~~~~~~~~~~~~~~" << endl;

       cin >> n;

       switch(n) {
           case 0: cout << "Bye!" << endl; break;
           case 1: com.sortBySum(); break;
           case 2: com.sortByOcc(); break;
       }
       cout << endl;

   } while(n != 0);

}

int main() {
```

```
    int drawNum;
    cout << "Enter the number of draw:" << endl;
    cin >> drawNum;

    Combination com(/****/);

    display(com);

    return 0;
}
```

Example 1:

```
Enter the number of draw:
2
~~~~~~~~~~~~~~~~~
0 exit
1 sort by sum
2 sort by occurrence
~~~~~~~~~~~~~~~~~
1
1 occurrence(s): 12
2 occurrence(s): 11
3 occurrence(s): 10
4 occurrence(s): 9
5 occurrence(s): 8
6 occurrence(s): 7
5 occurrence(s): 6
4 occurrence(s): 5
3 occurrence(s): 4
2 occurrence(s): 3
1 occurrence(s): 2

~~~~~~~~~~~~~~~~~
0 exit
1 sort by sum
2 sort by occurrence
~~~~~~~~~~~~~~~~~
2
6 occurrence(s): 7
5 occurrence(s): 8
5 occurrence(s): 6
4 occurrence(s): 9
4 occurrence(s): 5
3 occurrence(s): 10
3 occurrence(s): 4
2 occurrence(s): 11
2 occurrence(s): 3
1 occurrence(s): 12
1 occurrence(s): 2

~~~~~~~~~~~~~~~~~
0 exit
1 sort by sum
2 sort by occurrence
~~~~~~~~~~~~~~~~~
0
Bye!

Memory is released
```

Example 2:

```
Enter the number of draw:
4
~~~~~~~~~~~~~~~~~
0 exit
1 sort by sum
2 sort by occurrence
~~~~~~~~~~~~~~~~~
1
1 occurrence(s): 24
4 occurrence(s): 23
10 occurrence(s): 22
20 occurrence(s): 21
35 occurrence(s): 20
56 occurrence(s): 19
80 occurrence(s): 18
104 occurrence(s): 17
125 occurrence(s): 16
140 occurrence(s): 15
146 occurrence(s): 14
140 occurrence(s): 13
125 occurrence(s): 12
104 occurrence(s): 11
80 occurrence(s): 10
56 occurrence(s): 9
35 occurrence(s): 8
20 occurrence(s): 7
10 occurrence(s): 6
4 occurrence(s): 5
1 occurrence(s): 4

~~~~~~~~~~~~~~~~~
0 exit
1 sort by sum
2 sort by occurrence
~~~~~~~~~~~~~~~~~
2
146 occurrence(s): 14
140 occurrence(s): 15
140 occurrence(s): 13
125 occurrence(s): 16
125 occurrence(s): 12
104 occurrence(s): 17
104 occurrence(s): 11
80 occurrence(s): 18
80 occurrence(s): 10
56 occurrence(s): 19
56 occurrence(s): 9
35 occurrence(s): 20
35 occurrence(s): 8
20 occurrence(s): 21
20 occurrence(s): 7
10 occurrence(s): 22
10 occurrence(s): 6
```

```
4 occurrence(s): 23
4 occurrence(s): 5
1 occurrence(s): 24
1 occurrence(s): 4

~~~~~~~~~~~~~~~~~
0 exit
1 sort by sum
2 sort by occurrence
~~~~~~~~~~~~~~~~~
0
Bye!

Memory is released
```

~~~ END ~~~
~~~