Assignment One (5% of the Total Course Grade)

Due Date: 31 October 2025, 23:59 PM

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

2. You can check as many times as you want before the deadline.

3. Only a small set of the test cases are visible for the testing, and a more comprehensive set of test cases will be used for grading. In other word, passing all the visible test cases does not mean that you can get the full mark for this question. Try your best to thoroughly check your program.

4. Hidden test cases will not be released.

5. The marking of each question is based on the percentage of the total test cases that your solution can pass. Please indicate your **development environment** to help us run and grade your code, including OS, compiler, and IDE. If your submitted solution leads to a **compilation error**, zero mark will be given to that question and no manual checking to your solution is provided in such a case.

6. No late submission will be accepted.

7. Plagiarism check will be performed.

8. You only need to use the material from Lectures 1 to 3. It is **NOT** necessary to include any other library, except <iostream> <iomanip>.

9. You need to check your solution and submit it on Canvas.

Q1 [25%]

Write a program to print the pattern made up of two hollow triangles. The program receives an integer as input and then prints the pattern based on the integer. This integer represents the number of rows occupied by a hollow triangle. Here are some notes:

a) The input integer value (**data type: int**) is required to be **no less than 3**.

b) You can assume that the input is always an integer. However, if the input is less than 3, such as "0", "1", "-2", the program repeats and asks the user to input again.

c) The input number in the test cases will not be very large. So, do not worry about very large inputs.

Example 1:

```
Enter an integer not less than 3:
3
The input is valid!
The pattern is
  *
 **
*****
  **
  *
```

Example 2:

```
Enter an integer not less than 3:
-1
Invalid value. Input again!
Enter an integer not less than 3:
0
Invalid value. Input again!
Enter an integer not less than 3:
4
The input is valid!
The pattern is
   *
  **
 * *
*******
   * *
   **
```

```
      *
```

Q2 [25%]

In cryptography, ciphertexts can be viewed as the encrypted result of a plaintext (plain messages) using an encryption key. One popular cipher works as follows.

1. A plaintext message is chosen as input, e.g., we choose “tellhimaboutme” as our plaintext.

2. A short, secret word is chosen as the key, e.g., we pick “cafe” as our key.

3. If the chosen key is shorter than the plaintext, we repeat it until the length matches. For example, with input and key in Step 1 and 2 respectively, the repeated key should be “cafecafecafeca”.

4. Each character of the plaintext is shifted by the alphabet number of the corresponding. Character in the key outputted in Step 3. (e.g., the alphabet number of character “a” is 1, “e” is 5). For example, the alphabet number of “c” is 3, so the first character “t” in the plaintext of the following example will be shifted by 3 (the ASCII code adds 3) and will be changed to “w”.

5. After shifting, if the ASCII code is even larger than that of “z”, the character starts from “a” again. For example, the alphabet order of “f” is 6. After “u” in the following example is shifted by 5, it becomes “z”, and the additional one shift makes the character be “a” finally.

Plaintext:	tellhimaboutme
Key:	cafecafecafeca
Ciphertext:	wfrqkjsfepaypf

Write a program to convert an inputted ciphertext back to the plaintext.

Hint: For simplicity reason, the key used in your solution is always “cityu”. We assume that the length of the input ciphertext is longer than the key and the input ciphertext is valid.

Hint: You can use “char input[1000] = {'\0'}” to store ciphertext and use “cin >> input” to get the input. We assume that this array is large enough.

Example 1:

```
Input ciphertext:
znvdglnpddqhit
Output plaintext:
webelieveinyou
```

Example 2:

```
Input ciphertext:
wqyzowjwjnwjlsorviqmrf
Output plaintext:
theattackstarttomorrow
```

Q3 [50%]

Develop an Employee Management System that allows a department to manage its employees in a company. The system should be able to hire new employees, display all employees, transfer an employee to another department by ID, and sort the employees by either ID or date of entry. We assume that the maximum number of employees in employee management system is fixed to 10, and the IDs of employees are unique.

Requirements:

1. Complete the definition of the **structure Date** to hold the day, month, and year as integers.
2. Complete the definition of the **structure Employee** to hold an integer ID, a double salary, and a Date.
3. Complete the definition of the **structure Company** to hold an array of Employee structures and an integer to keep track of the current number of employees in this department.
4. Implement the ***listOptions*** function to print the menu options for the system as shown in the following examples.
5. Implement the ***hire*** function to allow hiring a new employee to the company. This function should prompt the user for the unique employee ID, salary, and date (day, month, year). Ensure that the department does not exceed its maximum capacity (i.e., 10). The ID of the new employee will not conflict with the old ones.
6. Implement the ***show*** function to display all the employees in the department. Each employee should be displayed with the ID, salary, and date entered correctly.
7. Implement the ***transfer*** function to transfer an employee by ID and the employee information will be removed from this system. The system should be prompted to enter the ID of the employee to remove.
8. Implement the ***sort_id*** function that sorts the employee in ascending order based on their ID.
9. Implement the ***sort_salary*** and ***sort_date*** function that sorts the employee in ascending order based on their salary or date of entry, correspondingly. **Different from ID, the salary or date of two employees could be the same.** If two employees have the same salary or date, they should be further sorted by ID in ascending order
10. Implement the ***sort*** function to allow the system to choose how to sort the employees (by ID or by salary or by date).

Instructions:

- You are provided with a code framework and a partially implemented main function.
- Your task is to complete the missing parts of the code to make it functional as per the requirements.
- Do not change the provided constants and function signatures.
- You can assume all input will be valid and formatted correctly, e.g., when hiring a new employee, you can assume that the provided id is not same as any existing one.
- Two special cases: 1) when hiring an employee, after typing "1", if the number of current employees is already 10, this employee will NOT be added and display "Department is full"; 2) when showing employees, if there is no employee now, display "Department is empty".

Hint-1: You can use functions in the <iomanip> library to control your output format, such as setprecision, setw, setfill, etc., when you show the information of each employee.

Hint-2: We assume that the input salary is in the range of [10000, 99999], e.g., 16000. When showing the salary of each employee, always showing two decimal places, e.g., 16000.00.

```
// Code framework. You can copy the following code to your solution and
start with it.

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

const int N = 10;

struct Date {

};

struct Employee {

};

struct Company {

};

void listOptions() {

}

void hire(/* */) {

}

void show(/* */) {

}

void transfer(/* */) {

}

void sort_id(/* */) {

}

void sort_salary(/* */) {

}

void sort_date(/* */) {
}

void sort(/* */) {
    int n;
    cout << "1: Sort by ID" << endl;
    cout << "2: Sort by Salary" << endl;
    cout << "3: Sort by Date" << endl;
    cin >> n;
}

void init(Company *c) {
    c->employees[c->num].id = 4;
    c->employees[c->num].salary = 10500;
    c->employees[c->num].date.day = 15;
    c->employees[c->num].date.month = 1;
    c->employees[c->num].date.year = 2012;
    (c->num)++;

    c->employees[c->num].id = 2;
    c->employees[c->num].salary = 20550;
    c->employees[c->num].date.day = 15;
    c->employees[c->num].date.month = 2;
    c->employees[c->num].date.year = 2017;
    (c->num)++;

    c->employees[c->num].id = 3;
    c->employees[c->num].salary = 10500;
    c->employees[c->num].date.day = 20;
    c->employees[c->num].date.month = 2;
    c->employees[c->num].date.year = 2021;
    (c->num)++;

    c->employees[c->num].id = 1;
    c->employees[c->num].salary = 30100;
    c->employees[c->num].date.day = 20;
    c->employees[c->num].date.month = 2;
    c->employees[c->num].date.year = 2021;
    (c->num)++;
}

int main() {
    Company comp;
    comp.num = 0;

    init(&comp);

    int opt;

    return 0;
}
```

Example 1: Hire an employee

```
~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
1
Input Employee ID:
5
Input Salary:
16000
Input Date (dd mm yyyy):
21 12 2023

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
004 10500.00 15-1-2012
002 20550.00 15-2-2017
003 10500.00 20-2-2021
001 30100.00 20-2-2021
005 16000.00 21-12-2023

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
0
Bye!
```

Example 2: Transfer an Employee

```
~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
1
Input Employee ID:
6
Input Salary:
22500
Input Date (dd mm yyyy):
01 01 2025

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
3
Enter the ID to be transfered
6

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
004 10500.00 15-1-2012
002 20550.00 15-2-2017
003 10500.00 20-2-2021
001 30100.00 20-2-2021

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
0
Bye!
```

Example 3: Sort employees by ID

```
~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
004 10500.00 15-1-2012
002 20550.00 15-2-2017
003 10500.00 20-2-2021
001 30100.00 20-2-2021

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
4
1: Sort by ID
2: Sort by Salary
3: Sort by Date
1

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
001 30100.00 20-2-2021
002 20550.00 15-2-2017
003 10500.00 20-2-2021
004 10500.00 15-1-2012

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
0
Bye!
```

Example 4: Sort employees by salary

```
~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
004 10500.00 15-1-2012
002 20550.00 15-2-2017
003 10500.00 20-2-2021
001 30100.00 20-2-2021

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
4
1: Sort by ID
2: Sort by Salary
3: Sort by Date
2

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
003 10500.00 20-2-2021
004 10500.00 15-1-2012
002 20550.00 15-2-2017
001 30100.00 20-2-2021

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
0
Bye!
```

Example 5: Transfer non-existent employee

```
~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
2
004 10500.00 15-1-2012
002 20550.00 15-2-2017
003 10500.00 20-2-2021
001 30100.00 20-2-2021

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
3
Enter the ID to be transferred
7
The ID cannot be found

~~~~~~~~~Welcome!~~~~~~~~~~
0: Exit
1: Hire
2: Show
3: Transfer
4: Sort
~~~~~~~~~~~~~~~~~~~~~~~~~~~
0
Bye!
```

~~~ End ~~~
~~~