Assignment #3 Queue & Stack

Instructions

1. The assignment is submitted in groups of **maximum 3** students from the same **lab OR** same **TA.**
2. Deadline of submission is **29/4/2024**
3. The submission will be on Google classroom.
4. Your submission should include a single **cpp file**, named LabGroup_ID1_ID2_ID3 (ex.: S13_20220022_20220023_20220024.cpp).
5. No late submission is allowed.
6. No submission through e-mails.
7. No rar files
8. No exe file submission.
9. **In case of Cheating, you will get a negative grade whether you give the code to someone, take the code from someone/internet, or even send it to someone for any reason.**
10. You must write clean code and follow a good coding style including choosing meaningful variable names.
11. In case of wrong submission, wrong file extension/type, missing files, plagiarism, extra submitted files, wrong naming, the assignment will not be accepted and no correction for these mistakes is allowed, and you will lose your grade.

Task 1: Hospital Emergency Room

We'll augment the functionalities from **scratch** of the emergency room simulation by adding a display function to view the patients in the **priority queue** and a size function to determine the number of patients currently in the queue.

Implement the following functionalities:

Patient Class:

Define a class Patient with attributes such as name, age, condition, and priority.

Priority can be determined based on the severity of the patient's condition.
Conditions will be normal, urgent and critical.
Priority will be 3, 2 and 1.
critical = 3 = maximum priority
urgent = 2 = medium priority
normal = 1 = minimum priority

Priority Queue Implementation:

Implement a priority queue data structure to manage patient tasks.

Operations on queue should include the following:

1- Add Patient Function:

Implement a function add_patient to add a new patient to the priority queue.
Add the patient to the priority queue based on his priority/condition.

2-Process Patient Function:

Implement a function process_patient to process/remove the patient with the highest priority from the queue.
Extract the highest priority patient from the queue and display their details.

3-Update Patient Function:

This function modifies the priority of a given existing patient in the queue.

This function takes patient and new priority/condition of the patient.

In case priority increased(condition of patient get worst)(ex. from normal to urgent condition) will update condition of the patient and rearrange the priority queue based on this.(handling the way of rearrange is opened to student)

In case priority decreased(condition of patient got better)(ex. from critical to urgent condition) will update condition of the patient and rearrange the priority queue based on this.

In case priority entered is same nothing will happen but handle the in function using a cout.

4-Display Function:

Implement a function display_queue to display the current patients in the priority queue.
This function should print out the details of each patient in the queue, including their name, age, condition, and priority.

5-Size Function:

Implement a function queue_size to determine the number of patients currently in the priority queue.
This function should return the size of the priority queue.

6-FrontName Function:

This function retrieves the name of the most urgent patient (the patient at the front of the priority queue) without removing them or altering the state of the queue.
It throws a string exception(cout) if the queue does not contain any patients.

The final output should include details of each patient processed, the current state of the priority queue (including patient details), and the number of patients in the queue. This extended simulation provides a more comprehensive view of the emergency room operations, allowing for better monitoring and management of patient tasks.

Note: Read input from the user, then display as the following sample output:

```
Patient Ahmed added to the queue.
Patient Mazen added to the queue.
Patient Omar added to the queue.
Front Patient Name: Ahmed
Current Patients in the Queue:
Name: Ahmed, Age: 35, Condition: critical, Priority: 3
Name: Mazen, Age: 45, Condition: urgent, Priority: 2
Name: Omar, Age: 25, Condition: normal, Priority: 1
Processing patient...
Processing patient: Ahmed (Priority: 3)
Current Patients in the Queue:
Name: Mazen, Age: 45, Condition: urgent, Priority: 2
Name: Omar, Age: 25, Condition: normal, Priority: 1
Priority/Condition of patient Ahmed updated to urgent
Current Patients in the Queue:
Name: Ahmed, Age: 35, Condition: urgent, Priority: 2
Name: Mazen, Age: 45, Condition: urgent, Priority: 2
Name: Omar, Age: 25, Condition: normal, Priority: 1
```

Task 2: Variable Processing System

We'll enhance the functionalities of a variable processing system by adding features to read C++ code from a file, extract variable details, and manage them using a stack-based approach.

Implement the following functionalities:

Variable struct:

Define a struct **Variable** with attributes **name**, **datatype**, and **value**.
Variables will be extracted from C++ code lines.

Code Processor Implementation:

Implement a code processor to read C++ code from a file and process variable declarations and assignments.

Your program will create a **stack of variables** and the operations of your code processor should include the following:

1. Process File Function:

- Read C++ code from a file named **"code.txt"** and process each line.
- Detect the end of the code to indicate completion and if the end of the code is reached, pop all variables from the stack and make it empty.

2. Process Line:

- The file starts with the main header **int main()** and ends with **return 0;**
- Each line in the file is either:
    1- A variable declaration: such as double a;
    2- A variable declaration and initialization: such as char ch='D';
    3- A variable assignment: such as x=1;
    4- End of program as in return 0;

- For each line of the code:
    1- If the line is a declaration, add the variable to the stack.
    2- If the line is an assignment, look for the variable in the stack and update its value.
    3- If the program ends the function display Stack should be called.

Assume that the datatypes available are : **int**, **double**, **float** or **char** only

4. Display Stack Function:

- Implement a function to display the current variables stored in the stack.
- Print details of each variable, including its name, type, and value.

Implement your stack class based on the STL list class. Do not use the STL Stack.

INPUT FILE EXAMPLE:

```
int main()
{
    int x=9;
    int y=10;
    double w=32;
    y=90;
    w=50;
    char c='r';
    float r=3.6;
    c='t';
    r=5.2;
    double p=7.15;
    return 0;
}
```

OUTPUT :

```
Stack Contents:
Name: x, Type: int, Value: 9;

Stack Contents:
Name: y, Type: int, Value: 10;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: w, Type: double, Value: 32;
Name: y, Type: int, Value: 10;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: w, Type: double, Value: 32;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: c, Type: char, Value: 'r';
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: r, Type: float, Value: 3.6;
Name: c, Type: char, Value: 'r';
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;
```

```
Stack Contents:
Name: r, Type: float, Value: 3.6;
Name: c, Type: char, Value: 't';
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: r, Type: float, Value: 5.2;
Name: c, Type: char, Value: 't';
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;

Stack Contents:
Name: p, Type: double, Value: 7.15;
Name: r, Type: float, Value: 5.2;
Name: c, Type: char, Value: 't';
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;

Reached return 0;
Stack Contents:
Name: p, Type: double, Value: 7.15;
Name: r, Type: float, Value: 5.2;
Name: c, Type: char, Value: 't';
Name: w, Type: double, Value: 50;
Name: y, Type: int, Value: 90;
Name: x, Type: int, Value: 9;
```

Queue Grading Criteria	
Adding patient	7.5
Proccess patient	7.5
Update Patient Function.	10
Display front	5
Display size	5
Display queue	5
Main	5
Output	5
Total	50

Stack Grading Criteria	
Process File Function	10
Process Variable Function	15
Display Stack Function	10
Main	5
Output	10
Total	50