National University Of Computer and Emerging Science
Lahore Campus

Object Oriented Programming

Assignment # 2

Due Date: Sunday, May 8th by 11:55pm..

Dear students we will be using auto-grading tools, so failure to submit according to the below format would result in zero marks in the relevant evaluation instrument.

1. Combine all your work in one folder. The folder must contain only .cpp files (no binaries, no exe files etc.).
2. Run and test your program on a lab machine before submission.
3. Rename the folder as ROLL-NUM_SECTION (e.g. 20i-0001_A) and compress the folder as a zip file. (e.g. 20i-0001_B.zip). do not submit .rar file.
4. Submit the .zip file on Google Classroom within the deadline.
5. Submission other than Google classroom (e.g. email etc.) will not be accepted.
6. The student is solely responsible to check the final zip files for issues like corrupt file, virus in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.
7. Displayed output should be well mannered and well presented. Use appropriate comments and indentation in your source code. Five Bonus marks will be awarded to well commented/indented code (for all questions).
8. Total Marks: 100
9. If there is a syntax error in code, zero marks will be awarded in that part of assignment.
10. Your code must be generic

Task Manager

You will write a **WorkList class** that manages a list of Task. The WorkList class manages a dynamically allocated array of Task objects. The **Task class** is provided for you in task.h and task.cpp, which could be modified(or you can make new one). Your WorkList class will be implemented in two files WorkList.h and WorkList.cpp. The class will support two basic functions:

1. Managing the WorkList. This includes adding and deleting tasks.
2. Starting task from the task list. This includes keeping track of which is the next task to be start. Obviously we will not actually be starting any task. "Starting" the task will consist of printing the name.

To support these functions, your WorkList class should have the following public member functions:

- The class should have a default constructor that builds an empty WorkList with space for 2 Tasks.
- **void AddTask(const Task s);**
  This function will add s to the end of the task list. This function needs to adjust the task array size when necessary. This function should not affect the current task index (for starting task). A task has following properties:
  1. Task Name.
  2. Total Time
  3. Task Type (Internal or External)
- **bool DeleteTask(const Task s);**
  This function tries to delete s from the task list. If s is in the list, the function deletes the Task from the list and returns true. Otherwise, nothing is done and the function returns false. If the task list has multiple copies of one Task, you should just delete one copy. This functions needs to adjust the Task array size when necessary to make sure that the number of Tasks in the list is at least half the size of the capacity. If after a Task is deleted, the WorkList object becomes empty, the function must make the object identical to the one created by a default constructor. This function should reset the current Task index only if the index is out of bound after a Task is deleted.
- **void ShowAll() const;**
  This function will print all of the Tasks in the task list on the screen.
- **void Start(int num=1);** This function "Start" num Tasks from the list. The Tasks in the list should be start in a circular manner: when reaching the end of the list, repeat from the beginning.
- **void ShowStatus() const;** This function shows the status information of the WorkList including the total slots allocated for the Task list (size of the Task array), the number of Tasks in the list, and the current Task index.
- The class should have a destructor that deletes the memory allocated for the WorkList when the WorkList object is deallocated.

The WorkList class must also overload the following operators and implement the described functionality accordingly:

1. **Addition operator (+):** adding a Task to a WorkList (i.e. WorkList object + Task object) This routine will return a new WorkList object. The returned WorkList will be a concatenation of all Tasks in the caller's WorkList and the single Task at the end (same as the result of AddTask).
2. **Subtraction operator (-):** removing a Task from a WorkList (i.e. WorkList object – Task object).This function will return a new WorkList object that has the same content as the result of DeleteTask.

All data members in this class should be private. To support the public and friend functions, the WorkList must maintain a number of data members. The most important one is the WorkList that stores an array of Tasks. However, since there is no size limit on the task list, it must be declared as a Task pointer (e.g. Task *plist;) so as to be used as a dynamically allocated array. The dynamic array must expand and shrink to ensure that its size is no more than twice that number of Tasks in the list or no more than 2. More specifically, there are three situations:

- At initialization, the size of the dynamic array should be 2.
- In the AddTask function, if the dynamic array is full (array capacity = number of Tasks), the array size needs to double to make room for new Tasks.
- In the DeleteTask function, if after a Task is deleted, the number of Tasks is less than ½ of the array capacity, the array size needs to be halved (with the exception that the capacity is 2 as in the initialized empty WorkList, in which case, the array should not shrink).

You should isolate two memory management functions in two routines: void doublesize() and void halfsize(). The taskes in both functions are similar with the following steps:

1. Allocate a new dynamic Task array of the desired size
2. copy the content in the old Task array to the new array
3. free the old array (plist)
4. set plist to the new array and set bookkeeping variables accordingly.
5. For diagnostic purposes, you must call ShowStatus() at the beginning of each function and before each function returns (see the behavior in the sample executable). Doublesize will be called in the AddTask function while halfsize will be called in the DeleteTask function.

The class must also contain other private member for bookkeeping with at least three private data members: an int member to record the capacity of the dynamic array, an int member to record the number of Tasks in the dynamic array, and an int member to record the current Task index (to support the Start function). These members are displayed in the ShowStatus function.

Test

A test program menu.cpp is provided. This program will allow you to interactively test the features of your WorkList class. Keep in mind that this program does not test everything that your class must handle. For example, it doesn't test the operator overloads. You should write other tests to verify you implement your memory allocation and deallocation correctly.

Honor Policy

This task is a learning opportunity that will be evaluated based on your ability.**Plagiarized reports or code will get a zero**. If in doubt, ask the course instructor.