CPSC 2150 – Algorithms and Data Structures II

Assignment 1: Task Scheduler Simulation

Total - 100 Marks

You should spend 80 percent of your time on solving the problem and only 20 percent for programming.

Learning Outcomes

- Evaluate and select appropriate data structures (SLL, DLL, CLL) based on operation requirements.
- Understand and implement the Iterator ADT for list traversal.
- Apply recursion to solve complex problems like dependency checking.
- Use stacks for expression processing and queues/priority queues for task scheduling.
- Analyze time and space complexity to make informed design decisions.

Resources

- Chapter 2, 3, 4, 5 and 6 of the textbook

Description

The task scheduler mimics systems used in project management tools, operating systems, and workflow automation, giving you experience with concepts used in industry. This is a hands-on design and development assignment like Jira or Trello that challenge your skills and boost your resume with a standout project! Dive in and showcase your ability to tackle real-world problems with creativity and expertise!

Design and implement a Task Scheduler that manages tasks with different priorities and deadlines. You will analyze and select the best data structures (from Singly Linked List (SLL), Doubly Linked List (DLL), Circular Linked List (CLL), or vector) and algorithms for efficient task management, perform complexity analysis, and use recursion, stack, queues, and priority queues to process tasks. The system will simulate scheduling tasks based on priority and deadlines, with an iterator to traverse the task list.

Requirements

1 **Task Class**

1.1 Define a Task class with attributes:

- taskID (string): Unique identifier.
- description (string): Task description.
- priority (int): Priority level (1 = highest, 10 = lowest).
- deadline (int): Deadline in hours from now (e.g., 24 means due in 24 hours).
- dependency (string): ID of another task this task depends on (empty if none).

1.2 Methods: Constructor, getters, and a method to display task details.

2 Linked List Selection and Implementation

2.1 Choose one of SLL, DLL, CLL or vector to store tasks. Justify your choice in a report (**answers.pdf**).

2.2 Implement the chosen linked list with:

- Node structure (with Task object and appropriate pointers).
- Methods: `insertTask(Task task)`, `deleteTask(string taskID)`, `findTask(string taskID)`.

2.3 Implement an Iterator ADT to traverse the list:

- Class TaskIterator with methods `hasNext()`, `next()`, and `reset()` to iterate through tasks.
- Use the iterator to display tasks or process them in order.

3 Task Scheduler Logic

3.1 Use a Priority Queue (implemented using a sorted-list or unsorted list) to schedule tasks based on priority and deadline.

- Tasks with higher priority (lower number) ~~and earlier deadlines~~ are processed first.
- If a task has a dependency, it cannot be scheduled until the dependency is completed.

3.2 Use a Stack to evaluate a dependency expression (e.g., a task's priority may be defined as a post/pre/infix expression like 3 + 2 * priority of dependency. See an example in section 6).

- For simplicity, convert infix expressions (e.g., 3 + 2 * 5) to postfix using a stack-based algorithm.
- Evaluate the postfix expression to compute the effective priority.

3.3 Use a Queue to store completed tasks in the order they are finished.

4 Recursive Component

4.1 Implement a recursive function to check for cyclic dependencies in the task list (e.g., Task A depends on B, B depends on C, C depends on A).

- Return true if a cycle exists, false otherwise.

4.2 Use recursion for a method like `findDependentTasks(string taskID)` to list all tasks that depend on a given task.

See **Implementation Details** for more detail.

5 Main Program

Create a menu-driven program with the following options:

- Add a task (prompt for ID, description, priority, deadline, dependency).
- Delete a task by ID.
- Display all tasks to be done using the iterator.
- Schedule the next task (use priority queue, check dependencies, update completed tasks queue).
- Check for cyclic dependencies (using recursion). Show the dependency list.
- Save finished tasks to a file
- Load tasks to be done from a file.
- Exit.

Sample menu:

```
Task Scheduler
1. Add Task
2. Delete Task
3. Display All Tasks
4. Schedule Next Task
5. Check Cyclic Dependencies
6. Save to File
7. Load from File
8. Exit
Enter choice:
```

6 File Handling

6.1 Save tasks to a text file such as tasks.txt in a format like:

```
T001,Write Report,5,24,,,
T002,Research,2,48,,,
T003,Review Notes,4,12,T001,,
T004,Submit Assignment,6,36,T002,,
T005,Prepare Presentation,2 + 1.5 * priority of T002,18,T003,,
```

Note that the expression **`2 + 1.5 * priority of T002`** for T005 requires the program to evaluate T002's priority (2) and compute 2 + 1.5 * 2 = 5. As it is shown the priority of a task can be represented by an expression that needs to be evaluated. You must use stack to evaluate those expressions.

6.2 Load tasks from the file, reconstructing the linked list.

7 Complexity Analysis Report (answers.pdf)

Submit a short report (1-2 pages) that:

7.1 Justifies your choice of linked list (SLL, DLL, CLL or vector) for this application, considering operations like insertion, deletion, and iteration.

7.2 Analyzes the time and space complexity of:

- Linked list operations (insertTask, deleteTask, findTask).
- Iterator operations (hasNext, next).
- Priority queue operations (enqueue, dequeue).
- Stack-based expression conversion and evaluation.
- Recursive dependency checking.

7.3 Discuss trade-offs (e.g., why you chose SLL over DLL or CLL, or why a sorted-list or unsorted-list for the priority queue).

Implementation Details

- Use `#include <iostream>, <string>, <fstream>, <vector>` (for priority queue), and other standard libraries.
- Ensure proper memory management (e.g., delete nodes in the linked list destructor).
- Handle edge cases (e.g., invalid task IDs, cyclic dependencies, file errors). The following shows the requirements for handling cycles:
  - **Detect Cycles:** Use recursion to identify if a task's dependency chain loops back to itself.
  - **Prevent Scheduling**: Tasks in a cycle should not be added to the priority queue until the cycle is resolved (e.g., by removing a task or breaking the dependency). You must remove the task with the least priority to break the cycle.
  - **User Feedback:** Report cycles to the user via the "Check Cyclic Dependencies" menu option, identifying tasks involved in cycles.
  - **Robustness:** Ensure the scheduler checks for cycles before adding tasks or scheduling them to avoid deadlocks.
- Validate inputs (e.g., priority between 1-10, non-negative deadlines).
- Use a stack for expression conversion/evaluation and a queue for completed tasks.
- Implement the priority queue (lower priority number = higher priority).
- You can reuse your own classes that are implemented in your previous lab exercises (such as SLL, DLL, or CLL). Please self-reference if you are reusing your own library.

- Use templates for Node to write your classes such as lists (SLL, DLL or CLL), stack, queue or priority-queue.

Grading Criteria

- Linked List Implementation (20 points): Correct SLL/DLL/CLL with insert/delete/find operations.
- Iterator ADT (15 points): Functional iterator for traversing tasks.
- Recursion (15 points): Correct cyclic dependency check and recursive dependent task listing.
- Stack and Expression Handling (15 points): Correct infix-to-postfix conversion and evaluation.
- Queue & Priority Queue (15 points): Proper task scheduling based on priority/deadline.
- File Handling and UI (15 points): Robust file I/O and user-friendly menu with error handling.
- Complexity Analysis Report (20 points): Clear justification of data structure choice and accurate complexity analysis.

Submission

Submit a zip file named **StudentNumber-Asgn1.zip** including all source code files (.cpp and .h), **answers.pdf**, and **README** file that explaining how to compile or run the program. For example, if your student number is 10023449, the submitted zip file must be named as **10023449-Asgn1.zip**.