Github Repo : [Link]

Team Stallions

202301165 Shrey Shah(Leader)

202301157 Parv khetawat

202301128 Abhishek kothari

202301032 Jainil jagtap

P11: Remainder and Task Scheduler

You are to build a Task scheduler with Remainder functionality,in which user are allowed to manage task based on deadline or importance level. Here we have use file handling concept which is used to store task in file ,we also include different functionality like adding of task, deletion of task marking task as completed,viewing completed task,and displaying task in order of priority and giving remainder of task with highest priority .

Description of our function:

1)addtask: In this function User is used to input the details of task (includes description,priority,deadline),and adds it to a file ('Task_List.txt'), and insert it into the priority queue.
2)FiletoQueue: This is used to load any left task from file into the priority queue at the start of the program.
3)remainderfunc:Calculates and displays the remaining days left for the task with highest priority.
4)deletetask:Deletes a specified task from both the file and the priority queue.

5)getdeadline:Retrive the remaining days left for a specific task.
6)markTask:Marks a task as completed, moving it to a separate file ('Tasks_completed.txt') and removing it from the original file and priority queue.
7)viewcompletetask:Display all completed task from the 'Tasks_completed.txt'file.
8)missingTask: Add the missing status to that task in (Task_List.txt) file that have passed their deadline,removes it from priority queue.

We are using this specific data structure :

Linked List :

- Used to implement the priority queue.
- Importance: The linked list is used to maintain the tasks in priority order based on their deadline and priority level. Each node contains information about the task such as description, priority, and deadline.

Priority Queue :

- Implemented using a linked list.
- Importance: The priority queue is used to store tasks based on their priority and deadline. Tasks are added to the priority queue in such a way that the task with the highest priority and nearest deadline is always at the front. This allows efficient retrieval and management of tasks based on their urgency.

Additionals :

Standard Library Containers (std::string, std::ifstream, std::ofstream):

- Used for various purposes such as storing task descriptions, reading/writing to files, etc.
- Importance: These containers provide convenient ways to manipulate strings and perform file I/O operations, which are essential for implementing features like adding tasks, reading/writing tasks to/from files, etc.

Time-Related Functions (std::time, std::localtime):

- Used to calculate the remaining days until the deadline of a task.
- Importance: These functions allow the program to determine the remaining days until the deadline of a task, which is crucial for prioritizing tasks in the priority queue based on their urgency.

#define Macro (IgnoreNewChar):

- Used for ignoring newline characters in input.
- Importance: Helps in handling user input by ignoring any newline characters left in the input buffer, ensuring clean input processing.

Pseudocode of the TASK MANAGER

1>Push function in class priority queue

```
function push(d, p, dy, m, y)
   temp = new Node(d, p, dy, m, y)
   now = time(0)
   ltm = localtime(&now)
   curr_year = 1900 + ltm->tm_year
   curr_month = 1 + ltm->tm_mon
   curr_day = ltm->tm_mday

   remaining_days = (y - curr_year) * 365 + (m - curr_month) * 30 +
(dy - curr_day)
```

```
    if head is null or (((head->year - curr_year) * 365 + (head->month - curr_month) * 30 + (head->day - curr_day)) > remaining_days) or ((((head->year - curr_year) * 365 + (head->month - curr_month) * 30 + (head->day - curr_day))) is equal to remaining_days and p > head->priority)
        temp->next = head
        head = temp
    else
        start = head
        while start->next is not null and ((((start->next->year - curr_year) * 365 + (start->next->month - curr_month) * 30 + (start->next->day - curr_day)) < remaining_days) or ((((start->next->year - curr_year) * 365 + (start->next->month - curr_month) * 30 + (start->next->day - curr_day)) is equal to remaining_days) and start->next->priority > p))
            start = start->next
        temp->next = start->next
        start->next = temp
    endif
Endfunction
```

- **Time complexity: O(n)**
- **Space complexity: O(1) (for creating temporary variables)**

2>Peek Function

```
function peek()
    if head is not null
        return head->data
    return "Priority queue is empty"
Endfunction
```

- **Time complexity: O(1)**

- **Space complexity: O(1)**

3>Pop function

```
function pop()
    if head is null
        return
    temp = head
    head = head->next
    delete temp
Endfunction
```

- **Time complexity: O(1)**
- **Space complexity: O(1)**

4>Isempty function

```
function isEmpty()
    return head is null
Endfunction
```

- **Time complexity: O(1)**
- **Space complexity: O(1)**

5>Displaying the priority queue

```
function printPQ()
    temp = head
    if temp is null
        print "No tasks in the Task manager"
    else
    while temp is not null
        print temp->data
        temp = temp->next
endfunction
```

- **Time complexity: O(n)**
- **Space complexity: O(1) (excluding space for the output)**

6>Function to add a task to the file and priority queue

```
Function addTask(fileName, pq)
    fileOut = openFileForWriting(fileName, appendMode)
    newTask = Task()

    output "Enter the task description:"
    readLineFromConsole(newTask.description)

    output "Enter the priority ranging in (0-9):"
    readIntegerFromConsole(newTask.priority)

    output "Enter the deadline date as dd-mm-yyyy:"
    readLineFromConsole(newTask.date)

    if fileOut is open then
        writeLineToFile(fileOut, newTask.description + "," +
newTask.priority + "," + newTask.date)
        closeFile(fileOut)
        output "Task successfully added to the file."
    else
        output "Unable to open file to write task."
    endif
    pq.push(newTask.description, newTask.priority,
parseInteger(substring(newTask.date, 0, 2)),
parseInteger(substring(newTask.date, 3, 2)),
parseInteger(substring(newTask.date, 6, 4)))
End Function
```

- **Time complexity: O(n)**
- **Space complexity: O(1) (excluding space for input variables)**

7> Function to load any left tasks from the file to priority queue

```
Function FileToQueue(pq, fileName)
    file = openFile(fileName)
    if file is open then
        try
            while not endOfFile(file) do
                pq.push(createTaskFromLine(readLineFromFile(file)))
            end while
            closeFile(file)
            output "Task successfully transferred from file to priority
queue."
        catch error
            output "Error processing file:", error
        end try
    else
        output "Unable to open file for file to queue transfer."
    end if

function createTaskFromLine(line)
    parts = splitLine(line, ',')
    return
        description: parts[0],
        priority: parseInt(parts[1]),
        dueDate: parseDueDate(parts[2])

function parseDueDate(dateStr)
    parts = splitLine(dateStr, '-')
    return
        day: parseInt(parts[0]),
        month: parseInt(parts[1]),
        year: parseInt(parts[2])
   End Function
```

- **Time complexity: O(n^2)**
- **Space complexity: O(1) (excluding space for input variables)**

8> function which implements remainder functionality, it is called automatically everytime the program runs

```
function remainderfunc(pq, fileName)
    a = pq.peek()
    fileIn = openFileForReading(fileName)

    if fileIn is open then
        while not endOfFile(fileIn) do
            s = readLineFromFile(fileIn)
            x = ""
            j = 0
            while s[j] is not ',' do
                x += s[j++]
            end while

            if x equals a then
                i = 0
            while s[i++] is not ',' do { } // Empty loop to skip to the next
comma
            i++
            while i is less than size of s and s[i++] is not ',' do { } //
Empty loop to skip to the next comma

            if i is less than size of s then
               dd = parseInteger(substring(s, i + 1, 2))
               mm = parseInteger(substring(s, i + 4, 2))
               yy = parseInteger(substring(s, i + 7, 4))
               now = getCurrentTime()
               remaining_days = (yy - getYear(now)) * 365 + (mm -
getMonth(now)) * 30 + (dd - getDay(now))
               output "Days remaining for most important task:",
remaining_days
               exit while
            end if
         end if
      end while
      closeFile(fileIn)
   else
      output "Unable to open file for reading."
   end if
end function
```

- **Time complexity: O(n^2)**
- **Space complexity: O(1) (excluding space for input variables)**

9> function to delete a given task from file and priority queue

```
function deletetask(s, fileName, pq)
   fileIn = open(fileName)
```

```
newFileOut = open("temp.txt")

if fileIn is open and newFileOut is open
    while getline from fileIn into x
        des = ""
        i = 0
        for i to x.size()
            if x[i] is ','
                break
            else
                des += x[i]
                prio = x[i + 1] - '0'
                i = i + 3
        dat = ""
        for j from i to x.size()
            dat += x[j]
        if des is s
            continue
        else
            write des, prio, dat to newFileOut

    close fileIn and newFileOut
    remove old file and rename new one
    print "Task successfully deleted from the file."
else
    print "Unable to open file for task deletion."

// Priority Queue Operations
head = pq.getHead()

if head is null
    print "Priority queue is empty. Can't delete the task."
    else
        if head->data is s
            pq.pop()
        else
            temp = head
            while temp->next is not null and temp->next->data is not s
                temp = temp->next
            if temp->next is not null
                tempNext = temp->next
                temp->next = tempNext->next
                delete tempNext
            else
                print "Task not found in the priority queue."
End function
```

- **Time complexity: O(n^2)**
- **Space complexity: O(1) (excluding space for input variables)**

10> function to get deadline of a task inputted by user

```
function getdeadline(a, pq, fileName)
    fileIn = open(fileName)

    if fileIn is open
        while getline from fileIn into s
            x = ""
            j = 0
            while s[j] is not ',' and j < s.size()
                x += s[j++]

            if x is equal to a
                i = 0
            while s[i] is not ',' and i < s.size()
                i++
            i++;
            while i < s.size() and s[i] is not ','
                i++

            if i < s.size()
                dd = stoi(s[i + 1 to 2])
                mm = stoi(s[i + 4 to 2])
                yy = stoi(s[i + 7 to 4])

                now = time(0)
                ltm = localtime(&now)
                curr_year = 1900 + ltm->tm_year
                curr_month = 1 + ltm->tm_mon
                curr_day = ltm->tm_mday

                remaining_days = (yy - curr_year) * 365 + (mm -
curr_month) * 30 + (dd - curr_day)
                return remaining_days
        close fileIn
    else
        print "Unable to open file for reading."
    endif
Endfunction
```

- **Time complexity: O(n^2)**
- **Space complexity: O(1) (excluding space for input variables)**

11> function to view all the completed tasks

```
function viewcompletedtask()
    fileIn = openFileForReading("Tasks_completed.txt")
    if fileIn is open then
        while not endOfFile(fileIn) do
            output readLineFromFile(fileIn)
        endwhile
        closeFile(fileIn)
    else
        output "Unable to open file for reading completed tasks."
    endif
end function
```

- **Time complexity: O(n)**
- **Space complexity: O(1) (excluding space for output variables)**

12>function to add missing tasks

```
function missingTask(fileName, pq)
    fileIn = openFileForReading(fileName)
    fileOut = openFileForWriting("temp1.txt", overwriteMode)

    if fileIn is open then
        while not endOfFile(fileIn) do
            s = readLineFromFile(fileIn)
            des = ""
            i = 0
            while s[i] is not ',' do
                des += s[i++]
            prio = parseInt(s[i + 1])
            i += 3
            dat = ""
            j = i
            while j < size(s) and s[j] is not ',' do
                dat += s[j++]
            if getdeadline(des, pq, fileName) < 0 then
                if s[j + 1] is not ',' then
                    writeLineToFile(fileOut, des + "," + prio + "," + dat +
",missing")
                else
                    writeLineToFile(fileOut, des + "," + prio + "," + dat)
                closeFile(fileIn)
            else
                output "Unable to open file for reading."
        end function
```

- **Time complexity: O(n^2)**
- **Space complexity: O(1) (excluding space for input variables)**

13>function to mark a task done

```
function markTask(fileName, pq)
    completedTask = ""
    output "Enter the completed task:"
    IgnoreNewChar
    readLineFromConsole(completedTask)

    myFile = openFileForWriting("Tasks_completed.txt", appendMode)

    if myFile is open then
        writeLineToFile(myFile, completedTask)
        closeFile(myFile)
    else
        output "Unable to open file for writing completed tasks."
    endif
    deletetask(completedTask, fileName, pq)
end function
```

- **Time complexity: O(n^2)**
- **Space complexity: O(1) (excluding space for input variables)**