**Final C++ Group Project**

Cale Stevens, Artem Fedorov, Ryan Revels-Scholte, John Winsor

Centre of Geographic Sciences

PROG 2100

Sharla Rolfe-Hunter

December 7th, 2023

**Contents**

[IPO Chart 3](#__RefHeading___Toc335_395012926)

[Important Information 4](#__RefHeading___Toc3642_395012926)

[Short Form IPO Charts 5](#__RefHeading___Toc342_395012926_Copy_1)

[*inputFromFile()* 5](#__RefHeading___Toc367_395012926)

[*addEntry()* 5](#__RefHeading___Toc369_395012926)

[*removeEntry()* 5](#__RefHeading___Toc371_395012926)

[*outputSchedule()* 6](#__RefHeading___Toc373_395012926)

[*sortList()* 6](#__RefHeading___Toc375_395012926)

[Flowchart 7](#__RefHeading___Toc342_395012926)

[Hierarchy Chart 8](#__RefHeading___Toc344_395012926)

[Runtime Screen Capture 8](#__RefHeading___Toc346_395012926)

## IPO Chart

|  |  |  |
| --- | --- | --- |
| IPO Worksheet for PROG 2100 Final Project | | |
| Problem Statement | Track a list of tasks (name, due date, priority), then sort then according to due date and priority (priority has priority over due date). Tasks should be able to be added and deleted. A final task list should be saved to an output file. | |
| Output | Sorted list of tasks | |
| Input | Individual tasks, previous task list | |
| Process | Notation |  |
| Diagram | See hierarchy chart section |
| Approach - Functional Requirements | 1. Ask user if they would like to import an existing schedule from a file. If so, ask the user for the filename and import the file. Loop through each line in the file and add each task to the main task list. If the user does not wish to import an existing schedule, use an empty task list.  2. Ask the user if they would like to (1. Add an entry, 2. Delete an entry, 3. Output the schedule).  3. If they select option 1, Prompt the user for the task name, due date, and priority. Add a new task to the main task list using this information.  4. If the user selects option 2, display a list of all tasks and prompt the user to input the number corresponding task. Remove the specified task from the main task list.  5. If the user selects option 3, loop through each task in the main task list and sort them into three smaller lists based on priority (one list for priority 1, one list for priority 2, etc.). Then loop through each smaller list (starting at priority 3) and add each task back to the main task list in order of due date (closest first).  6. Output the main task list to a file with each task on a separate line and each task attribute separated by a tab character.  7. Ask the user if they would like to continue the program. If yes, then go back to step 2. |
| Solution | See flowchart section | |
| Desk Check |  | |
| Comments & Reflection |  | |

## Important Information

// Example entry in schedule file. Each attribute is separated by a tab

1 Task A 2023-12-05 3

// This is the struct that holds a task.

struct Task {

std::string name;

std::string date;

int priority;

}

// How to add an element to the task list

std::map<int, task> taskList; // Creates the list

Task task; // Create a struct

task.name = "Task A"; // Specify the task name

task.date = "2023-12-05"; // Specify the task date. Please use the ISO 8601 format

task.priority = 2; // Specify the task priority. Integer between 1 and 3.

taskList[5] = task; // Add the created task at index 5

// Remove element

taskList<int, Task>::iterator iterator; // Basically a reference to a map element

iterator = taskList.begin(0); // Move the iterator to the beginning of the map (index 0)

iterator += 5; // Move the iterator to the sixth element (index 5)

taskList.erase(iterator); // Delete the task at the index pointed to by iterator

## Short Form IPO Charts

### *inputFromFile()*

|  |  |
| --- | --- |
| Input | Filename  Empty task list |
| Process | 1. Ask the user for the filename and import the file.  2. For each line in the file, loop through it and look for tabs or newline characters.  3. If a tab or newline character is found, save everything before that (excluding what has already been saved) as an attribute of the current task. Use a counter to track what attribute is expected.  4. Once all attributes of the task have been located, create a new entry in the main task list with the attributes found. |
| Output | Filled task list |
| Notation | Filename = name of file to access |

### *addEntry()*

|  |  |
| --- | --- |
| Input | Task name  Task due date  Task priority |
| Process | 1. Ask the user for the name, due date and priority of the task to create.  2. Create a new task entry with the inputted information. |
| Output | New task |
| Notation | Priority is an integer between 1 and 3.  Task name should be a string  Task due date should be in the ISO 8601 format (YYYY-MM-DD). |

### *removeEntry()*

|  |  |
| --- | --- |
| Input | Task ID  Main task list |
| Process | 1. Display a list of all tasks in the main task list  2. Prompt the user to input the ID of the task they would like to remove.  3. Remove the task with the inputted ID |
| Output |  |
| Notation | Task ID is an integer |

### *outputSchedule()*

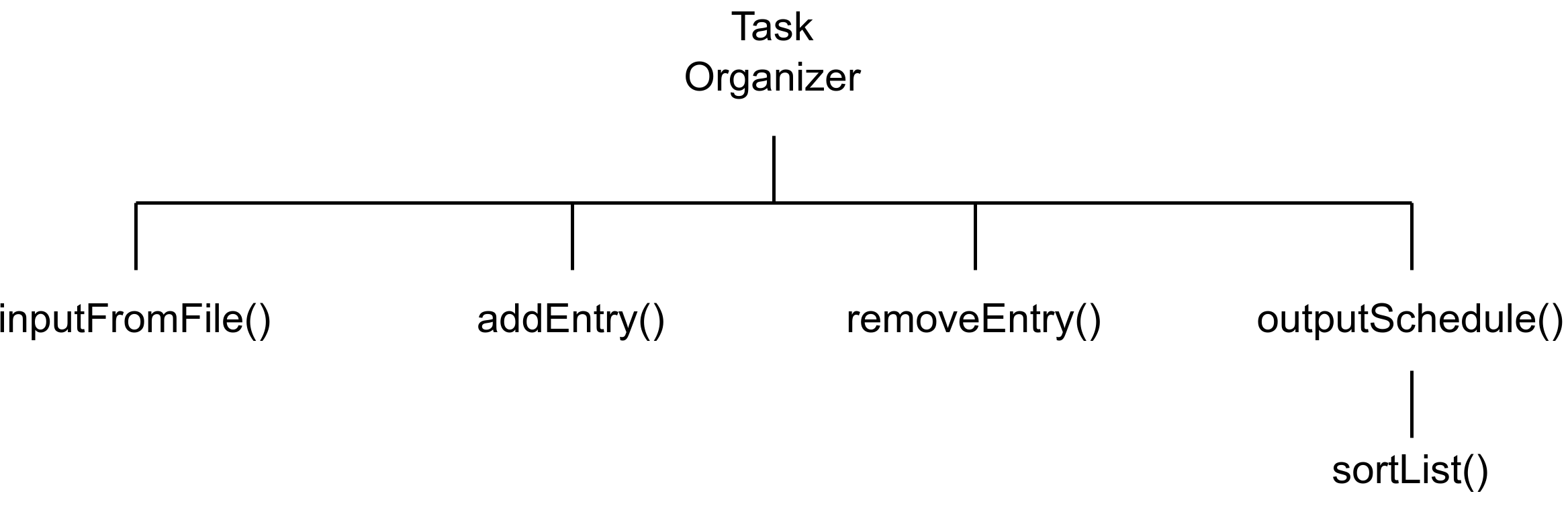
|  |  |
| --- | --- |
| Input | Main task list |
| Process | 1. Loop through each task in the main task list and sort them into three smaller lists based on priority (one list for priority 1, one list for priority 2, etc.). Remove each task as it is added to a priority list.  2. Then loop through each smaller list (starting at priority 3) and add each task back to the main task list in order of due date (closest first). Use sortList() to accomplish this.  3. For each entry in the main task list, append a line to the output file with each attribute in the entry separated by tabs. |
| Output | File with schedule information |
| Notation |  |

### *sortList()*

|  |  |
| --- | --- |
| Input | Unsorted list  Main task list |
| Process | 1. Loop through the list and add each task back to the main task list in order of due date (closest first).  3. For each entry in the main task list, append a line to the output file with each attribute in the entry separated by tabs. |
| Output | Sorted list (main task list) |
| Notation | The main task list is the main list holding all tasks. |

## Flowchart

## Hierarchy Chart

****