Scheduler

Pawan Sutar | pawansutar@outlook.com

Overview- This is a console app to generate work schedule for a week by assigning different types of installers to work on different types of buildings.

Task Overview
Given a list of buildings and list of employees, generate the schedule for the coming work week
Create a minimum viable solution by focusing on basic functionality and correctness

Solution Mechanism
This solution has been prepared by implementing a rather straightforward approach similar to any Greedy algorithms. I spent some time researching about linear programming algorithms that can be implemented to represent this problem however used a straightforward approach considering the time constraints. The file structure is as described below.

```
Scheduler
|
|--- scheduler.h
|                    - Header file with class methods and free functions declarations
|--- scheduler.cpp
|                    - Class methods and free function definitions
|--- main.cpp
                     - Entry point of the app, main function calls free and class functions to produce result
```

(This structure was chosen due to the time limit and the code can be organized better into multiple modules.)

Input
vector<Employee> employees : List of different types of employees
vector<Building> buildingsList : List of different types of buildings
(These vectors are generated in the main function for simplicity and due to time constraints)

Output (Printed on console, not returned)
map<int, vector<int>> result : Hashmap to store building IDs as keys and list of employees assigned to work on that building as values

Classes
Employee
The class to generate particular installer/worker instances
Private members- ***ID, authorization, availability***
Public methods- ***getID(), getAuthorization(), getAvailability(), setAvailability()***

Building
The class to generate particular building instances
Private members- ***ID, type, status, priority***
Public methods- ***getID(), getType(), getStatus(), setStatus(), getPriority()***

Major Free functions

Function	Parameters	Returns	Description
assign()	Result map, Building instance, employees queue and an integer	void	Updates the building ID as key in result map and an array of assigned employees as values, changes the status of buildings to "assigned"
checkRequired()	Queues of different types of employees and a Building instance	bool	Checks if we have enough employees with required authorization to work on the passed building
swapPriority()	Queue of buildings	void	If there are not enough employees to work on a building, this function ignores the priority, removes and reinserts that particular building in the queue and moves on with next building in the queue

Solution Mechanism

- The input lists i.e. the lists of employees and buildings are generated in the main functions
- The employees list is split into queues of employees based on their authorization
- The buildings list is converted to a queue of buildings assuming the list is sorted based on the priority
- The program iterates over an enum containing days of the week from Monday to Friday
- For each day, the program iterates over the queue of buildings
- If the building status is "assigned for work" it skips the building
- If the building status is "unassigned", it checks the building type and verifies if there are enough employees with required authorization to work on that building and assigns employees to the building
- The assignment is represented by a map, that holds building IDs as keys and arrays/vectors of employees to work on that building as values
- if there are not enough employees available for the day, the building is removes from the queue and reinserted into the queue to be scheduled on further days of the week

Bottlenecks and Improvements

- When there are not enough employees available to work on the building for a day, the building is dequed and then enqued to the buildings queue. I used this approach considering the time constraints, but it can lead to moving a high-priority building to the end of queue. A better implementation is possible to move buildings back in the queue without compromising their priority.
- The main function may look too long in this submission but another approach would be to define a Schedule class and add functionalities to perform the operations performed in the main function.
- The solution focuses on utilizing the workforce and not keeping most workers idle. Better approach can be possible to utilize the workforce, ensure fair worker assignment using round robin approach.

Solution Screenshot

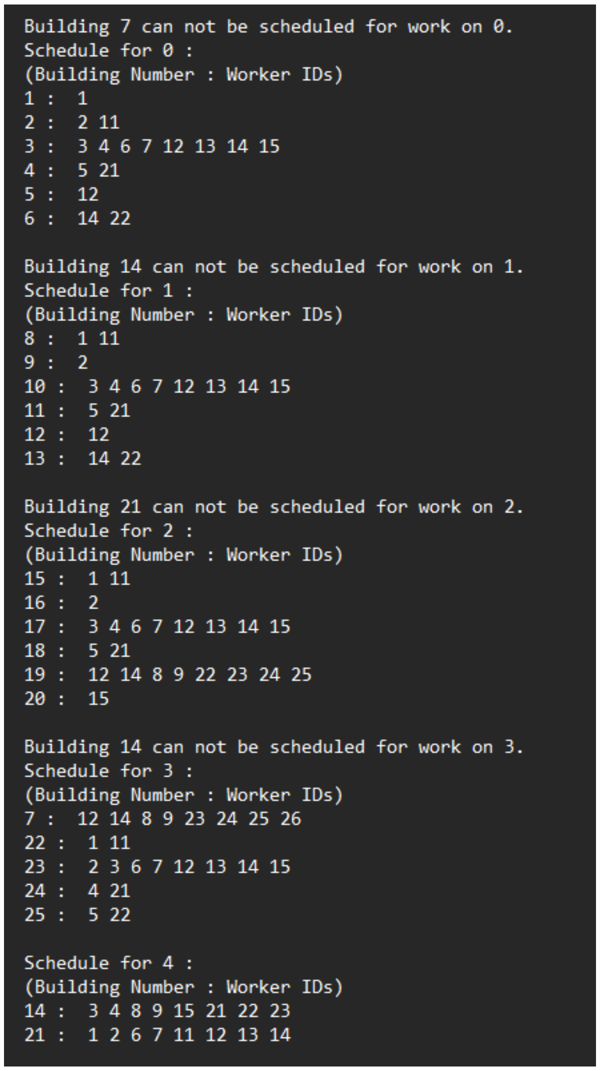
The days from Monday to Friday are represented by 0 to 4 respectively

enum days {Monday = 0, Tuesday = 1, Wednesday = 2, Thursday = 3, Friday = 4 };

Format

Building ID 1 : { Employee ID 1, Employee ID 2, Employee ID 3 }

Employee IDs 1, 2 and 3 will work on Building ID 1



```
Building 7 can not be scheduled for work on 0.
Schedule for 0 :
(Building Number : Worker IDs)
1 :  1
2 :  2 11
3 :  3 4 6 7 12 13 14 15
4 :  5 21
5 :  12
6 :  14 22

Building 14 can not be scheduled for work on 1.
Schedule for 1 :
(Building Number : Worker IDs)
8 :  1 11
9 :  2
10 :  3 4 6 7 12 13 14 15
11 :  5 21
12 :  12
13 :  14 22

Building 21 can not be scheduled for work on 2.
Schedule for 2 :
(Building Number : Worker IDs)
15 :  1 11
16 :  2
17 :  3 4 6 7 12 13 14 15
18 :  5 21
19 :  12 14 8 9 22 23 24 25
20 :  15

Building 14 can not be scheduled for work on 3.
Schedule for 3 :
(Building Number : Worker IDs)
7 :  12 14 8 9 23 24 25 26
22 :  1 11
23 :  2 3 6 7 12 13 14 15
24 :  4 21
25 :  5 22

Schedule for 4 :
(Building Number : Worker IDs)
14 :  3 4 8 9 15 21 22 23
21 :  1 2 6 7 11 12 13 14
```