SPL – Assignment 1
Medical Warehouse managment
2024/Spring

Teaching assistant: Or Kadosh

Lecturer: Gil Einziger

Submission Date: 19/05/24

1. Before you start

It is mandatory to submit all the assignments in pairs. If you can't find a partner, use the designated forum.

● Read the assignment together with your partner and make sure you understand the tasks. Before writing any code or asking questions, make sure you read the whole assignment.

● Skeleton files will be provided on the assignment page, you must use these classes as a basis for your project and implement (at least) all the functions that are declared in them.

● For any request for ease or postponing the deadline for submitting, please contact the responsible lecturer.

Please Notice:
While you are free to develop your project on whatever environment you want, your project will be tested and graded ONLY on a CS LAB UNIX machine.
It is your own responsibility to deliver a code that compiles, links and runs on it.
Failure to do so will result in a grade 0 to your assignment.
Therefore, it is mandatory that you compile, link and run your assignment on a lab unix machine before submitting it.
We will reject, upfront, any appeal regarding this matter! We do not care if it runs on any other Unix/Windows machine.
Please remember, it is unpleasant for us, at least as it is for you, to fail your assignments, just do it the right way.

2. Assignment Goals

The goal of this assignment is to develop an object-oriented system and enhance your practical skills in C++ through the use of classes, standard data structures, and distinctive C++ features like the "Rule of 5." You will acquire knowledge on managing memory in C++ and preventing memory leaks. The program you create should be optimized for maximum efficiency.

SPL – Assignment 1
Medical Warehouse managment
2024/Spring

3. Assignment Definition

In the fictional region of NevrLand, a devastating natural disaster has disrupted normal life, affecting countless communities.
In response to the urgent need for medical supplies across the region, a non-profit organization has established an emergency distribution system operated by dedicated volunteers.

This assignment involves developing a C++ program that simulates the management of a medical supplies distribution system. The program will manage a central warehouse, assign volunteers and beneficiaries, distribute medical supplies, and perform other related actions, as described later.

Upon starting, your program will open the warehouse, organize resources, assign volunteers, and prepare to service requests for medical supplies. The program should begin by announcing the warehouse is operational with a message: "Medical services are now open!" Following this, the system will wait for user commands to process actions in a loop, managing the distribution of supplies efficiently and effectively.

SPL – Assignment 1
Medical Warehouse managment
2024/Spring

3.1 Program Flow

The program accepts the path of a configuration file as its first command-line argument:

string configurationFile = argv[1];

The configuration file sets the initial state of the warehouse, detailing the existing beneficiaries and volunteers. Once the program is initiated, it sets up the warehouse based on this configuration file and then begins the simulation by invoking the **start()** method of the **MedicalWarehouse** class, followed by printing to console: “Medical services are now open!".

User Interaction: The program then enters a loop, waiting for user input to direct its operations. Each command triggers a specific action, such as handling supply requests, updating resource allocations, or modifying volunteer assignments. After executing an action, the program awaits further instructions.

3.2 Classes Overview

This system is structured around several core classes that facilitate the operation of the medical supplies distribution system. Each class is designed to handle specific aspects of the distribution process, from managing volunteers to processing requests.

MedicalWarehouse: Acts as the central hub for the emergency supplies distribution. holds a list (Vector) of volunteers. The class manages:

- **Pending Requests**: A queue of supply requests that have been made but not yet processed.
- **In Process Requests**: Requests that are currently being handled by volunteers.
- **Completed Requests**: Successfully fulfilled requests where the supplies have reached the beneficiaries.
- Additionally, the class manages unique identifiers for volunteers and beneficiaries and keeps track of whether the warehouse is operational with the isOpen flag.

Beneficiary: Represents entities that require medical supplies, such as hospitals and clinics. Each beneficiary has a unique ID which links them to their supply requests throughout the simulation. There are 2 types of such entities:

- Hospitals
- Clinics

Volunteer: An abstract base class for various volunteer roles within the warehouse. Each volunteer type has specific responsibilities:

- **Inventory Manager**: Prepares and organizes supplies based on requests.
  This volunteer will provide the supply to the driver.

- **Courier**: Responsible for the physical delivery of supplies to beneficiaries. Attributes include maxDistance they can cover and distancePerStep, determining how fast they can deliver supplies.

SupplyRequest: Details an individual request for supplies. It includes:

- Request Status: Tracked with an enum indicating stages such as PENDING, COLLECTING, ON THE WAY, and DONE.
- Linked Volunteers: References to the Inventory Manager and Courier involved in the request.
- Beneficiary ID: Links the request to the beneficiary who made the request.

Here are some more information about the supply status:

• PENDING – when a medical entity places an request it should be stored in the warehouse until one of the inventory manager will handle it.
• COLLECTING – when one of the inventory managers has been associated with the request, the status of the request changes from pending to collecting.
• ON THE WAY – when one of the Couriers has been associated with the request, the status of the request changes from collecting to delivering.
• DONE – when the Beneficiary gets his package (Courier finishes processing the request), the status of the request changes from ON THE WAY to DONE.

Core Action: An abstract class used to define actions that can be performed within the system, such as adding a supply requests or advancing the simulation. Each action:

- **Act()**: A pure virtual function that executes the action on the MedicalWarehouse.
- **toString()**: Returns a string representation of the action.
- **Action Status**: Indicates whether the action has been completed successfully or encountered an error.

3.3 Actions

Advance Simulation - This core action advances the simulation by moving it forward by a specified number of time units. Each unit represents a step in managing the distribution of medical supplies from the warehouse to beneficiaries.

Simulation Scheme

The action consists of four main stages:

1. **Supply request Processing**:
    - Review all the requests in the **PendingRequests** list of the warehouse and update their statuses based on the availability of volunteers:
        - **Pending request** are assigned to Inventory Managers for collection.

- **Collecting requests** are handed over to Courier for delivery.
- request s remain in the **InProcessRequests** during volunteer processing. This step ensures that request statuses are only updated when a volunteer is actively handling them, avoiding scenarios where a
- request might change status without available volunteer support.

- Note: Supply requests are associated with volunteers based on their availability and role compatibility. The system is designed to prioritize older requests to prevent "request starvation," where newer requests are handled before older ones still waiting.

2. **Advance Time Units**:
    - Decrement the **timeLeft** for each Inventory Manager by one for each simulation step.
    - Decrease the **distanceLeft** for each Courier according to their **distancePerStep**. Ensure that **distanceLeft** does not drop below zero to accurately represent the distance remaining to the delivery location.
3. **Volunteer and Supply Requests Completion Check**:
    - At the end of each step, check all volunteers for completed tasks:
        - If a Inventory Manager finishes collecting, the request's status updates and moves to the delivery queue.
        - If a Courier completes a delivery, the request is moved to the **CompletedRequests** list, marking it as fulfilled.

Additional Considerations

- Ensure that all calculations respect logical constraints, such as non-negative distances. For instance, if a Transport Officer has a **distanceLeft** of 4 units but a **distancePerStep** of 6, then after one simulation step, **distanceLeft** should be adjusted to 0, not a negative number.

Command Syntax and Usage

- **Syntax**: **step <number_of_steps>**
- **Example**: **step 3**
- This action does not result in an error. Assume the **number_of_steps** provided is always a positive integer.

SPL – Assignment 1
Medical Warehouse managment
2024/Spring

AddRequset – Creates a new request.

- Syntax: request <beneficiary_id>
- Example: request 5
- Outcome: The request is initialized with a PENDING status and added to the PendingRequests list in the warehouse.
  This action results in an error if the provided beneficiary ID does not exist or if the beneficiary has reached their maxRequests limit: "Cant place this request."

RegisterBeneficiary – Adds a new healthcare facility or beneficiary to the warehouse system.

Syntax: register <beneficiary_name> <type> <distance> < maxRequests >

Example: register CityHospital hospital 5 200

Outcome: This action adds a new beneficiary and is not prone to errors under normal conditions.

PrintRequestStatus – Displays details about a specific supply request.

Syntax: requestStatus <request_id>

Example: requestStatus 4

Will print:

Request ID: 4

Status: Pending/Collecting/ON THE WAY/DONE

Beneficiary ID: <beneficiary_id>

Inventory Manager: <inventory_manager_id>/None
- None in case the order didn't reach the Inventory Manager stage.

Courier: <Courier_id>/None
- None in case the order didn't reach the Courier stage.

- Outcome: If the request ID does not exist, this action results in an error: "Request does not exist."

PrintBeneficiaryStatus – Provides detailed information about a beneficiary, including all requests made.

Syntax: beneficiaryStatus <beneficiary_id>

Example: beneficiaryStatus 9

Will print:

Beneficiary ID: 9

SPL – Assignment 1
Medical Warehouse managment
2024/Spring

Request ID: <request_id>

Status: Pending/Collecting/ON THE WAY/DONE

Request ID: <request_id> (**in case there are more than one supply requests)**

Status: Pending/Collecting/ON THE WAY/DONE

…

…

…

Requests Left: <num_requests_left>

- Outcome: This action results in an error if the beneficiary ID does not exist: "Beneficiary does not exist."

PrintVolunteerStatus – Shows detailed information about a volunteer's current status and workload.

Syntax: volunteerStatus <volunteer_id>

Example: volunteerStatus 5

Will print:

Volunteer ID: 5

IsBusy: True/False (Depending on whether the volunteer is currently assigned a task or not)

RequestID: <request_id>/None – None in case isBusy is False

TimeLeft: <time_left>/None –
(Depending on whether the volunteer is a Supply Manager/Courier/Doesn't process any request)

RequestsLeft: <request_left>

- Outcome: Results in an error if the volunteer ID does not exist: "Volunteer does not exist."

SPL – Assignment 1
Medical Warehouse managment
2024/Spring

PrintActionsLog – Lists all actions that have been performed by the user, excluding the current log action.

Syntax: log

Will print:

<action_1_name> <action_1_args> <action_1_status>

...

<action_n_name> <action_n_args> <action_n_status>

Example:

register CityHospital hospital 5 200 COMPLETED

request 0 COMPLETED

request 0 COMPLETED

request 0 ERROR (Reached maxRequests limit)

advanceSimulation 1 COMPLETED

- Outcome: This action never results in an error.

CloseSystem – Concludes operations, prints all requests with their status, and terminates the program.

Syntax: close

Will print:

Request ID: <request_1_id>, Beneficiary ID: <beneficiary_1_id>, Status: <request_1_status>

...

Request ID: <request_n_id>, Beneficiary ID: <beneficiary_n_id>, Status: <request_n_status>

- Outcome: Ensures that all memory is freed before exiting to prevent memory leaks, and never results in an error.

BackupWarehouse – Saves all current state information of the warehouse.

Syntax: backup

Instructions: To use a global variable in a file, include the declaration :
extern 'MedicalWarehouse* backup;'

- Outcome: Updates the backup with the latest state of the warehouse. Overwrites any previous backups and does not result in an error.

SPL – Assignment 1
Medical Warehouse managment
2024/Spring

RestoreWarehouse – Restores the warehouse to the last saved state from a backup: (warehouse itself, Beneficiaries, volunteers, supply requests, and actions history).

Syntax: restore

- Outcome: If called before any backup has been performed, it results in an error: "No backup available."

3.4 Input File Format

The input file establishes the initial state of the medical warehouse system, containing data organized line-by-line in the following order:

1. **Beneficiaries** – Each line describes a healthcare facility or beneficiary in the following pattern:

Syntax: beneficiary <beneficiary_name> <facility_type> <location_distance> <max_requests>

Examples:

- beneficiary CityHospital hospital 4 50 // CityHospital is a hospital, located 4 distance units away, with a maximum of 50 supply requests allowed.
- beneficiary CommunityClinic Clinic 3 30 // CommunityClinic is a clinic, 3 distance units away, with a maximum of 30 supply requests.

Volunteers – Each line provides details about a volunteer, specifying their role and operational parameters:

Syntax: volunteer <volunteer_name> <volunteer_role> <cooldown_or_maxDistance> (depend if the volunteer is an Inventory Manager or a Courier)<distance_per_step>

Examples:

- volunteer Sarah inventory manager 2 // Sarah is a Inventory Manager with a cooldown period of 2 time units between tasks.
- volunteer John courier 12 5 // John is a Courier, , can travel a maximum distance of 12 units, moving 5 units per step.

Note: Ensure that each volunteer's attributes are parsed correctly, reflecting their specific roles. For example, an Inventory Manager might have a cooldown time, while a Courier would have attributes related to travel distance and steps per move.

This structured input will initiate your program, setting up the initial resources, beneficiaries, and volunteer staff, which will then interact through the simulation to efficiently manage and distribute medical supplies.

4. Provided Files in Skeleton.zip

For the Medical Supplies Distribution System project, the following header and source files are included within the skeleton.zip package to facilitate the initial setup of your programming environment:

- MedicalWarehouse.h
- SupplyRequest.h
- Beneficiary.h
- Volunteer.h
- Action.h
- main.cpp

Additionally, the following resources are provided to assist in your development process:

configFileExample.txt – An example configuration file that outlines the initial setup of the warehouse, including predefined beneficiaries and volunteers.

RunningExample.pdf – A document that illustrates expected outputs and operational flows based on example inputs, serving as a guide to how the system should function under various scenarios.

Implementation Requirements:

You are required to implement all functions as declared in the provided header files and ensure they fulfill their intended purposes according to their names and signatures. Modifications to the function signatures or the introduction of global variables are **strictly prohibited**, as such changes may lead to compilation errors and negatively impact your grade.

Rule-of-Five Considerations

Ensure that any class managing resources (e.g., dynamically allocated memory) implements all five special member functions where applicable: the destructor, copy constructor, copy assignment operator, move constructor, and move assignment operator.
This practice is crucial to prevent resource leaks and ensure proper resource management.

Avoid adding unnecessary Rule-of-Five implementations to classes that do not manage resources, to keep the code clean and efficient.

Following these guidelines will ensure that your project is robust, maintains resource integrity, and is aligned with the objectives of creating an efficient medical supplies distribution system.

5. submission Instructions

Your project should be submitted in a single zip file named “student1ID_student2ID.zip”.
Organize the files within your submission as follows to ensure clarity and correctness in evaluation:

src/: This directory should contain all source files (.cpp) used in the assignment.

include/: This directory should house all the header files (.h) related to your project.

bin/: Leave this directory empty as it will be used to store the compiled executable file. Do not include any binary files in your submission.

makefile: Include a makefile that automates the compilation of your source files into the bin/ directory and creates an executable named "medical_warehouse".

Compilation and Execution

Your project will be compiled and linked by executing the make command.

The executable will be tested by running it with various scenarios and input files, for example, bin/medical_warehouse rest/input_file.txt.

Ensure your code compiles without any warnings or errors on departmental computers to avoid compatibility issues.

Memory Leak Testing

Your program will be evaluated for memory management using VALGRIND with the following command:

valgrind --leak-check=full --show-reachable=yes bin/medical_warehouse [additional program parameters]

The expected output from VALGRIND should indicate that all memory has been properly freed and there are no leaks possible:

All heap blocks were freed -- no leaks are possible

ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

Compiler Flags

The compilation of your program must include the following flags to ensure all standards and checks are enforced:

-g -Wall -Weffc++ -std=c++11 -Iinclude

6. recommendation

Implement the Rule of Five Appropriately:
It is essential for the robustness and efficiency of your project to correctly implement the Rule of Five where necessary, particularly for classes managing dynamic resources.
Ensure that your classes are correctly managing memory with proper constructors, destructors, and assignment operators.
This is crucial for maintaining the integrity of the program and avoiding resource leaks, especially given the dynamic nature of managing medical supply requests and volunteer assignments.

Test Your Submission Thoroughly:
After uploading your project to the submission system, take the time to re-download and extract the files to verify their integrity.
It is imperative that you then compile and run the project on university lab computers to confirm that it operates as expected.
Failing to ensure that your project compiles and runs correctly on the department's systems could result in receiving a **zero** for the assignment.
This step is crucial to avoid compatibility issues and to ensure that your submission meets all technical requirements.

I WISH YOU GOOD LUCK AND DON'T FORGET TO ENJOY THE JOURNEY! 😊