Basics of Programming02

NAME – NICOLE BINTA-FUNMI LEKWOT
NEPTUN CODE – I8VP90
COURSE ID - (BMEVIIIAA03)
LAB GROUP ID - CS16D
LAB INSTRUCTOR – DR VAITKUS MÁRTON

PROJECT TITLE – HOSPIDITI (HOSPITAL CHATBOX)
PROJECT – FINAL PROJECT
SUBMISSION DATE – 13.05.25

Introduction

The Hospiditi Hospital Management System is a C++ application designed to facilitate booking appointments and answering some frequently asked questions, including department listings, doctor information, appointment scheduling, and availability checking.
It demonstrates object-oriented programming principles, polymorphism. It allows users to book appointments with doctors by the medical departments, check the hospital's operating hours, and manage appointment records through file storage.

Program Interface

The Hospiditi system operates through a command-line interface. When the program starts, it attempts to load existing appointment data from the "appointments.txt" file. The user interacts with the system by entering text commands, and the system responds with appropriate information or actions.

Starting the Program

The program can be run in the terminal.

User Commands

The system responds to various commands including:

- `help` – shows the main menu with all the possible commands

- `booking` – it starts the appointment booking process
- `doctors` – it list all doctors by department
- `departments` – it shows all available departments
- `opening hours` – it displays hospital opening hours
- `appointments` - list all scheduled appointments
- `exit` - terminate the program.

Terminating the Program

By typing "exit" the system automatically saves all appointment data to the "appointments.txt" file.

Program Execution

Initial Setup

When launched, the system attempts to load existing appointment data. If the file exists, appointments are loaded into memory. It starts with an empty list if there is no preexisting data.

Booking Process

When "book appointment" is selected, the system guides them through a series of prompts:

1. Enter patient name
2. Select a department from the displayed list
3. Choose a doctor from that department
4. Specify day of the week
5. Enter appointment time (HH:MM format)

The system validates each input, it checks if the department exists, if the doctor is available, and if the hospital is open at the said time. If yes, the appointment is confirmed and stored.

Appointment Management

Users can view all scheduled appointments using the "today's appointments" command. Each appointment shows the time, patient name, doctor, and department.

Department and Doctor Information

The "departments" command lists all available medical departments, while the "doctors" command shows all doctors organized by their respective departments.

Input and Output

Input

The system accepts text commands and information through standard input (keyboard). For appointment booking, the user provides:

- Patient name
- Department name (must match existing departments)
- Doctor name (must be in the selected department)
- Day of week (e.g., "Monday")
- Time (in HH:MM format)

Output

The system generates text responses to the console, including:

- Command confirmations
- Error messages
- Lists of departments, doctors, and appointments
- Appointment confirmations

File Format

Appointments are stored in a text file (appointments.txt) with the following format:
Count
Name; department; doctor; day; time

Count - number of appointments, followed by one line per appointment with fields separated by semicolons.

Program Structure

Class Hierarchy

Using inheritance:

- `ISerializable` - For saving/loading data
- `IHospital` - Base hospital functionality

- `Hours` - Hospital opening hours
- `Department` - Department information
- `DepartmentsInfo` - Department listing functionality
- `Appointments` - Appointment handling
- `HospitalInfo` - General hospital information47

Main Classes

Hospiditi
It implements all interfaces and provides the core functionality:

- Appointment management
- Department and doctor listings
- Hospital hours information
- Question processing

IDepartment
Implementation of the Department interface that stores:

- Department name
- List of doctors in the department5

Appointment
A structure that stores appointment details:

- Patient name
- Department
- Doctor
- Day
- Time

Data Structures

- The system uses dynamic arrays for appointment storage with automatic expansion
- Standard C++ containers (vectors) for departments and doctors
- String processing for user input and command interpretation
- Custom comparison operators for appointment equality checking

Memory Management

The system allocates memory for appointments dynamically and handles proper cleanup in the destructor. It includes exception handling for memory allocation failures and implements expansion of the appointments array when needed3.

Testing and Verification

The program includes safety mechanisms to ensure reliable operation:

- Input Validation: Checks for valid departments, doctors, and appointment times
- Exception Handling: The use of try-catch blocks around critical operations like file I/O and memory allocation
- Availability Checking: Ensures that doctors are available before booking appointments
- Opening Hours Validation: Ensures appointments are only booked during hospital operating hours
- Case-Insensitive Matching: Uses lowercase conversion for comparisons to avoid case-sensitivity issues

Improvements and Extensions

There are some improvements which could improve this system:

1. Appointment Cancellation: Allowing the cancellation or rescheduling of appointments
2. Calendar: Including a visual calendar view for appointments

Difficulties Encountered

I encountered multiple challenges including:

1. Input Validation: Ensuring user inputs are properly validated and processed especially implementing the lowercase.
2. File I/O: Managing file operations with proper error handling, my input initially didn’t save to the “appointments.txt” file

Conclusion

The Hospiditi Hospital Chatbox successfully implements a functional appointment booking system. It demonstrates object-oriented design principles through interface implementation and inheritance.
The command-line interface provides accessibility. The usage of error handling and input validation helps create a better user experience.