Kingdom of Saudi Arabia
Qassim University
College of Computer Science &
Information Technology

CS214 – Data Structures
Project Title:
Hospital Reservation System
Semester: 462

Submitted by:

- **[Rawan Alfouzan] (441203131)**
- **[Aljazi Aleqab] (441203328)**

Submitted to: Dr. Hoba Alsultan

Table of Contents

1. Data Structure Selection

Data Structure Used: Linked List

Justification:

- **Dynamic Data Handling:**

The hospital reservation system requires frequent additions and deletions of patient records and reservations. Linked lists provide efficient dynamic memory allocation and resizing compared to arrays.

- **Efficient Insertions and Deletions:**

With linked lists, inserting or deleting a patient or reservation at any position can be done without shifting elements, making it ideal for real-time data manipulation.

- **Simplified Traversal:**

It's easy to traverse patient data and update information using linked lists, which helps maintain and manage hospital records smoothly.

- **Separation of Data:**

Using separate linked lists for patients and their reservations simplifies data organization and access, which improves the structure and maintainability of the system.

2. Main Functions of the System

The Hospital Reservation System includes a set of core functions that ensure efficient management of patient records and appointment scheduling. Each function is designed to meet the system's operational requirements and maintain data integrity. Below is a detailed description of the main functions implemented:

1. registerPatient() – Register a New Patient
Registers a new patient by collecting personal details such as name, gender, and age then give each patient an unique ID. The function ensures data is properly formatted and stored within the system's linked list structure. This is a prerequisite for making any reservation.

2. addReservation() – Add a New Reservation
Schedules an appointment for a registered patient by selecting a department and entering the desired date and time.And give each reservation an unique ID.

Key Features:
• Validates that no other reservation exists in the same department at the same date and time.
• Ensures that the same patient cannot book more than one appointment at the same time, regardless of the department.
• Checks that the entered date and time are within acceptable ranges (e.g., month ≤ 12, hour ≤ 23).
• Charges a reservation fee of 150 SAR, which is required at the time of booking. This fee is recorded and deducted later when the patient completes the appointment and pays the remaining amount.
This function ensures schedule consistency and integrates payment tracking into the reservation process.

Continue..

3. cancelReservation() – Cancel an Existing Reservation
Removes a reservation from the system. This function allows staff to cancel previously scheduled appointments, freeing the reserved slot for future use. It ensures that canceled reservations are properly deleted and not shown in active records.

4. completeReservation() – Mark Reservation as Completed
Marks an active reservation as completed once the patient finishes their appointment.

Key Features:
• Confirms that the patient has paid the full amount, accounting for the 150 SAR reservation fee paid earlier.
• Updates the reservation status to “Completed” for proper tracking.
• Supports financial integrity by ensuring accurate billing.

5. displayReservations() – Display All Reservations
Displays all current reservations in the system with complete details including patient ID, department, time, and reservation status. This function helps staff monitor scheduled appointments and identify available slots.

6. deletePatient(patientId) – Delete a Patient Record
Removes a patient’s record from the system using their unique ID. The function also deletes all reservations associated with that patient to maintain consistency and avoid orphaned appointment data in the system.

3. C++ Class Implementation

Patient structure

```cpp
#include <iostream>
#include <string>
using namespace std;

// Structure to store patient information
struct Patient {
    int id;               // Unique patient ID
    string fullName;      // Full name of the patient
    string birthDate;     // Patient's date of birth
    string gender;        // Gender
    string maritalStatus; // Marital status
    string phone;         // Contact phone number
    Patient* next;        // Pointer to the next patient (linked list)
};
```

Reservation structure

```cpp
// Structure to store reservation information
struct Reservation {
    int id;                // Unique reservation ID
    string department;     // Department where the reservation is made
    string date;           // Date of the reservation
    string time;             // Time of the reservation
    string status;           // Status of the reservation (Confirmed, Completed, Cancelled)
    Reservation* next;       // Pointer to next reservation in the linked list
    int patientId;           // ID of the patient who made this reservation
    float totalCost;         // Total cost including base and extra charges
};
```

Hospital reservation system class :

Private members:

```
28  // Class to manage patients and reservations
29  class HospitalReservationSystem {
30  private:
31      Patient* patientHead;         // Head pointer for patients linked list
32      Reservation* reservationHead;// Head pointer for reservations linked list
33      int nextPatientId;            // Counter for assigning patient IDs
34      int nextReservationId;        // Counter for assigning reservation IDs
35
```

Public members:

Cunstructor:

```
36  public:
37      // Constructor initializes pointers and starting ID values
38      HospitalReservationSystem() {
39          patientHead = nullptr;
40          reservationHead = nullptr;
41          nextPatientId = 1000;
42          nextReservationId = 2000;
43      }
```

Function: RegisetrPatient( ):

```
// Register a new patient and store their information
void registerPatient() {
    Patient* newPatient = new Patient();       // Create new patient node
    newPatient->id = nextPatientId++;          // Assign patient ID and increment

    cout << "\nEnter patient's full name: ";
    cin.ignore();
    getline(cin, newPatient->fullName);        // Get full name with spaces

    while (true) {                              // Loop until valid birth date
        cout << "\nEnter birth date (e.g., 1990-01-01): ";
        cin >> newPatient->birthDate;
        cin.clear();
        cin.ignore(1000, '\n');

        // Check formatting and basic structure of date
        if (newPatient->birthDate.size() == 10 &&
            newPatient->birthDate[4] == '-' &&
            newPatient->birthDate[7] == '-' &&

            isdigit(newPatient->birthDate[0]) && isdigit(newPatient->birthDate[1]) &&
            isdigit(newPatient->birthDate[2]) && isdigit(newPatient->birthDate[3]) &&
            isdigit(newPatient->birthDate[5]) && isdigit(newPatient->birthDate[6]) &&
            isdigit(newPatient->birthDate[8]) && isdigit(newPatient->birthDate[9])) {


            int year = stoi(newPatient->birthDate.substr(0, 4));   // Extract year
            int month = stoi(newPatient->birthDate.substr(5, 2)); // Extract month
            int day = stoi(newPatient->birthDate.substr(8, 2));   // Extract day

            if (year >= 1900 && year <= 2100 && month >= 1 && month <= 12 && day >= 1 && day <= 31) {
                break; // Exit loop if date is valid
            }
        }
        cout << "\n❌ Invalid date format. Please enter birth date like 1990-01-01."; // Error message
    }
    int choice;

    // Loop to validate gender input
    do {
        cout << "\nEnter gender (1 for Female, 2 for Male): ";
        cin >> choice;

        if (choice != 1 && choice != 2) {
            cout << "\n❌ Invalid input. Please enter 1 for Female or 2 for Male.";
        }
    } while (choice != 1 && choice != 2);

    newPatient->gender = (choice == 1) ? "Female" : "Male"; // Assign gender based on input

    // Loop to validate marital status input
    do {
        cout << "\nEnter marital status (1 for Married, 2 for Single): ";
        cin >> choice;

        if (choice != 1 && choice != 2) {
            cout << "\n❌ Invalid input. Please enter 1 for Married or 2 for Single.";
        }
    } while (choice != 1 && choice != 2);

    newPatient->maritalStatus = (choice == 1) ? "Married" : "Single"; // Assign marital status based on input

    // Loop to validate phone number format
    do {
        cout << "\nEnter phone number (e.g., +9665xxxxxxxx): ";
        cin >> newPatient->phone;

        // Check if number starts with +966 and is 13 characters long
        if (newPatient->phone.substr(0, 4) != "+966" || newPatient->phone.length() != 13) {
            cout << "\n❌ Invalid phone number. It must start with +966 and be 13 characters long.";
        } else {
            break; // Valid phone number
        }
     } while (true);

    newPatient->next = patientHead; // Add patient to the beginning of the list
    patientHead = newPatient;

    cout << "\nNew patient registered with ID: " << newPatient->id << endl; // Show assigned patient ID
}

// Search for a patient by ID
Patient* findPatientById(int id) {
    Patient* current = patientHead;
    while (current) {
        if (current->id == id)
            return current; // Return the patient if ID matches
        current = current->next;
    }
    return nullptr; // Not found
}
```

Function: AddReservation( )

```
    // Start of function to create a reservation
    void addReservation() {
        int patientId;
        cout << "\nEnter patient ID for reservation: ";

        cin >> patientId;

        Patient* patient = findPatientById(patientId); // Search for patient by ID
        if (!patient) {
            cout << "\nPatient ID not found."; // Patient doesn't exist
            return;
        }

        Reservation* newRes = new Reservation(); // Create new reservation node

        newRes->patientId = patientId; // Link reservation to patient

        int departmentChoice;
cout <<"\nEnter department\n 1: Neurology\n 2: Cardiology\n 3: Dermatology\n 4: Pediatrics\n 5: General\n 6: Dentistry : ";
    cin >> departmentChoice;

    // Assign department name based on user choice
    switch (departmentChoice) {
        case 1: newRes->department = "Neurology"; break;
        case 2: newRes->department = "Cardiology"; break;
        case 3: newRes->department = "Dermatology"; break;
        case 4: newRes->department = "Pediatrics"; break;
        case 5: newRes->department = "General"; break;
        case 6: newRes->department = "Dentistry"; break;
        default:
            cout << "\nInvalid department choice!";
            delete newRes; // Cancel creation if department is invalid
            return;
    }
     string date;
     while (true) { // Loop until a valid date is entered
         cout << "\nEnter date in Gregorian (2025-MM-DD): ";
         cin >> date;
         cin.clear();
         cin.ignore(1000, '\n');

         // Validate date structure
         if (date.size() == 10 && date[4] == '-' && date[7] == '-' &&
             isdigit(date[0]) && isdigit(date[1]) && isdigit(date[2]) && isdigit(date[3]) &&
             isdigit(date[5]) && isdigit(date[6]) && isdigit(date[8]) && isdigit(date[9])) {

             int year = stoi(date.substr(0, 4));
             int month = stoi(date.substr(5, 2));
             int day = stoi(date.substr(8, 2));

             if (year >= 2025 && year <= 2100 && month >= 1 && month <= 12 && day >= 1 && day <= 31) {
                 newRes->date = date;  // Assign validated date
                 break;
             }
         }
         cout << "\n❌ Invalid date. Please enter a valid date like 2025-04-18."; // Error message for invalid date
     }

     while (true) { // Loop until valid time is entered
         cout << "\nEnter time (e.g., 14:30): ";
         cin >> newRes->time;
         cin.clear();
         cin.ignore(1000, '\n');

         // Validate time format
         if (newRes->time.size() == 5 && newRes->time[2] == ':' &&
             isdigit(newRes->time[0]) && isdigit(newRes->time[1]) &&
             isdigit(newRes->time[3]) && isdigit(newRes->time[4])) {

             int hour = stoi(newRes->time.substr(0, 2));
             int minute = stoi(newRes->time.substr(3, 2));
            if (hour >= 0 && hour <= 23 && minute >= 0 && minute <= 59) {
                break; // Valid time
            }
        }
        cout << "\n❌ Invalid time format or range. Please use HH:MM (e.g., 14:30) and valid time.";
    }
    Reservation* current = reservationHead;
    while (current != nullptr) {
        // Check if this patient already has a confirmed reservation at this date/time
        // OR if another reservation exists for the same department at same date/time
        if ((current->patientId == patientId && current->date == newRes->date &&
             current->time == newRes->time && current->status == "Confirmed") ||
            (current->date == newRes->date && current->time == newRes->time &&
             current->department == newRes->department && current->status == "Confirmed")) {

            cout << "\n❌ This time slot is already taken (either by same patient or department).";
            delete newRes;
            return;
        }
        current = current->next;
    }
    int confirmPayment;
cout << "\nReservation fee is 150 SAR.\nIt must be paid now to confirm the appointment.\nIt is non-refundable." << endl;
 cout << "\nDid the patient pay the reservation fee? (1 = Yes / 2 = No): ";
    cin >> confirmPayment;

    if (confirmPayment != 1) {
cout << "\nReservation was not created because the patient did not pay the required fee."; // Don't create if unpaid
        delete newRes;
        return;
    }
    newRes->totalCost = 150;                    // Set base reservation cost
    newRes->status = "Confirmed";               // Set status as confirmed
    newRes->id = nextReservationId++;           // Assign unique reservation ID
    newRes->next = reservationHead;             // Insert at head of reservation list
    reservationHead = newRes;
    cout << "\nReservation confirmed with ID: " << newRes->id << endl; // Confirmation message
}
```

Function: DisplayReservations( )

```
251     // Display all reservations with their details
252     void displayReservations() {
253         Reservation* current = reservationHead;
254         cout << "\n--- All Reservations ---\n";
255         while (current) {
256             cout << "Reservation ID: " << current->id
257                  << "\n | Patient ID: " << current->patientId
258                  << "\n | Department: " << current->department
259                  << "\n | Date: " << current->date
260                  << "\n | Time: " << current->time
261                  << "\n | Status: " << current->status
262                  << "\n | Bill: " << current->totalCost << " SAR"
263                  << endl;
264             cout << "-----------------------------------------\n";
265             current = current->next;
266         }
267 
268     }
```

Function: DeletePatient ( )

```
269     // Function to delete a patient and all their related reservations
270     void deletePatient(int patientId) {
271         Patient* currentPatient = patientHead;
272         Patient* prevPatient = nullptr;
273         while (currentPatient != nullptr) {
274             if (currentPatient->id == patientId) {
275                 Reservation* currentRes = reservationHead;
276                 Reservation* prevRes = nullptr;
277 
278                 while (currentRes != nullptr) {
279                     // Find all reservations linked to this patient and delete them
280                     if (currentRes->patientId == patientId) {
281                         if (prevRes == nullptr) {
282                             reservationHead = currentRes->next;
283                         } else {
284                             prevRes->next = currentRes->next;
285                         }
286                         Reservation* tempRes = currentRes;
287                         currentRes = currentRes->next;
288                         delete tempRes;
289                     } else {
290                         prevRes = currentRes;
291                         currentRes = currentRes->next;
292                     }
293                 }
294                 // Now delete the patient from the patient list
295                 if (prevPatient == nullptr) {
296                     patientHead = currentPatient->next;
297                 } else {
298                     prevPatient->next = currentPatient->next;
299                 }
300                 delete currentPatient;
301                 cout << "\nPatient and their reservations have been successfully deleted";
302                 return;
303             }
304             prevPatient = currentPatient;
305             currentPatient = currentPatient->next;
306         }
307         cout << "\n❌ Patient not found. Please make sure the Patient ID is correct";
308     }
```

Function: CancelReservation( )

```
310     // Cancel a reservation by ID
311     void cancelReservation() {
312         int resId;
313         cout << "\nEnter reservation ID to cancel: ";
314         cin >> resId;
315 
316         Reservation* current = reservationHead;
317         while (current) {
318             if (current->id == resId) {
319                 if (current->status == "Cancelled") {
320                     cout << "\nReservation is already cancelled."; // Already cancelled
321                     return;
322                 }
323                 if (current->status == "Completed") {
324                     cout << "\nCannot cancel a completed reservation."; // Can't cancel after completion
325                     return;
326                 }
327 
328                 current->status = "Cancelled"; // Set status to cancelled
329                 cout << "\nReservation cancelled successfully.";
330                 return;
331             }
332             current = current->next;
333         }
334 
335         cout << "\n❌ Reservation ID not found."; // No matching ID
336     }
```

Function: CompleteReservation( )

```
337     // Mark a reservation as completed and calculate additional cost
338     void completeReservation() {
339         int resId;
340         cout << "\nEnter reservation ID to complete: ";
341         cin >> resId;
342         Reservation* current = reservationHead;
343         while (current) {
344             if (current->id == resId) {
345                 if (current->status == "Completed") {
346                     cout << "\nReservation is already completed."; // Already marked done
347                 return;
348             }
349             if (current->status == "Cancelled") {
350                 cout << "\nCannot complete a cancelled reservation.";
351                 return;
352             }
353             cout << "\nReservation fee: 150 SAR (already paid and non-refundable)";
354             float additionalCost;
355             cout << "\nEnter additional cost of services (e.g., lab = 50 SAR, x-ray = 150 SAR , etc.): ";
356             cin >> additionalCost;
357             float remaining = additionalCost - 150; // Calculate remaining to be paid
358             cout << "\nRemaining amount to be paid by patient: " << remaining << " SAR\n\n";
359             int paidChoice;
360             cout << "\nDid the patient pay the remaining amount? (1 = Yes / 2 = No): ";
361             cin >> paidChoice;
362             if (paidChoice == 1) {
363                 current->totalCost += remaining; // Add additional cost to total
364                 current->status = "Completed";        // Mark as completed
365                 cout << "\nReservation marked as completed.";
366             } else {
367                 cout << "\nReservation status remains as: " << current->status << ".\nPayment not completed.";
368             }
369             return;
370         }
371         current = current->next;
372     }
373     cout << "\nReservation ID not found."; // No match found
374 }
```

The Main( ) Function:

```
int main() {
    HospitalReservationSystem system;
    int choice;

    do {
        // Display menu
        cout << "\n\n--- Hospital Reservation System ---\n";
        cout << "1. Register patient\n";
        cout << "2. Add reservation\n";
        cout << "3. Cancel reservation\n";
        cout << "4. Complete reservation\n";
        cout << "5. Display all reservations\n";
        cout << "6. Delete patient\n";
        cout << "7. Calculate daily income\n";
        cout << "8. Exit\n";

        cout << "Enter your choice (1-8): ";
        cin >> choice;

        if (choice < 1 || choice > 8) {
            cout << "\n❌ Invalid choice. Please enter a number between 1 and 8.\n"; // Input validation
            continue;
        }

        // Execute user choice
        switch (choice) {
            case 1: system.registerPatient(); break;
            case 2: system.addReservation(); break;
            case 3: system.cancelReservation(); break;
            case 4: system.completeReservation(); break;
            case 5: system.displayReservations(); break;
            case 6: {
                int patientId;
                cout << "\nEnter patient ID to delete: ";
                cin >> patientId;
                system.deletePatient(patientId);
                break;
            }
            case 7: {
                string date;
                cout << "\nEnter date (2025-MM-DD): ";
                cin >> date;
                system.calculateDailyIncome(date);
                break;
            }
        }

    } while (choice != 8); // Exit loop on user choosing 8

    return 0;}
```

4. Bonus Feature

Description:

This feature calculates the total income generated on specific date by summing up the full totalCost of all reservations made on that day.

Purpose:

- Provides a clear overview of the hospital's income for any given day.
- Supports financial reporting and helps in tracking daily earnings efficiently.
- Useful for administrative planning and decision-making.

Key Point:

Calculates the full amount for each reservation, including the initial fee and any additional charges.

Implementation:

Function: CalculateDailyIncome( )

```
375 void calculateDailyIncome(string date) {
376     Reservation* current = reservationHead;
377     float totalIncome = 0;
378     bool found = false;
379 
380     while (current != nullptr) {
381         if (current->date == date ) {
382             totalIncome += current->totalCost;
383             found = true;
384         }
385         current = current->next;
386     }
387 
388     if (found) {
389         cout << "\nTotal income for " << date << " is: " << totalIncome << " SAR\n";
390     } else {
391         cout << "\nNo reservations found on this date.\n";
392     }
393 }
394 };
```

References :

- Data Structures Through C++
  Author: Yashavant Kanetkar
  Edition: 3rd Edition

- CS214 Lecture Slides on Data Structures