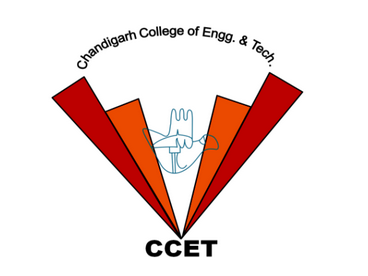
**CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)**

****

Government institute under Chandigarh (UT) Administration, affiliated to Punjab University, Chandigarh

Department of Computer Science & Engineering

**Semester**: CSE 3rd

**SUBJECT:** Data Structures Practical (CS351)

**Problem 4: Case Study of doubly Linked List**

**Submitted by: Submitted to:**

Bhavyam Dhand Dr. R.B. Patel

(CO23316) (Professor)

**INDEX**

|  |  |  |
| --- | --- | --- |
| S.No | Content | Page no. |
|  | OBJECTIVE | 3 |
|  | IMPLEMENTATION OF CODE | 4 |
|  | Add a scheme in Database: | 4 |
|  | Delete a scheme/subject from Scheme Database | 5 |
|  | Update a scheme/subject in Database | 5 |
|  | Print report card of a students | 6 |
|  | Print report of results complete for a semester: | 7 |
|  | Provide the provision to sort the list on any desired field e.g., Name | 7 |
|  | CODE IMPLEMENTATION | 8 |
|  | CODE RESULTS | 16 |

**Objective:**

**Case study of Doubly linked list:**

Write a complete program to implement students’ Examination Records of an

institute. Use doubly linked list for the implementation. You can use C/C++ language.

The attributes of a record is as follows.

* Enrollment Number/ Roll Number (Primary Key)
* Students Name
* Father’s Name
* Date of Birth
* Semester and year of admission
* Subject/Scheme of study

The attribute of Subject of a Semester

* Semester and year
* Subject Code (Should be received from database of scheme)
* Subject Title (Should be received from database of scheme)
* Maximum Marks for Sessional (Should be received from database of scheme)
* Maximum Marks for Theory (Should be received from database of scheme)
* Marks awarded in Sessional of theory/Practical
* Marks awarded in Theory/Practical

1. This program should perform the following operations. Design an Input panel to manage database system.

* Add a scheme in Database
* Delete a scheme/subject from Scheme Database
* Update a scheme/subject in Database
* Print report card of a students
* Print report of results complete for a semester
* List record/result (Random access) and also provide the provision to sort the list on any desired field e.g., on Name, Enrollment Number, etc.

1. Measure the performance of your system on which this code is implemented by measuring the following
   1. Number of nodes possible to create on the machine
   2. Time required to search a required
   3. Time required to delete a required
   4. Time required to insert a required
2. Plot a graph for number of nodes vs. Time required to create list of given number of nodes.
3. Further program should maintain database in files (scheme file), students
4. record/information file. Processed result in separate file.

**IMPLEMENTATION OF CODE:**

* + 1. **Add a scheme in Database:**
       1. **Psuedocode:**

FUNCTION addRecord(start)

1 PRINT "Adding a New Record..."

1 INPUT Enrollment Number

n WHILE Enrollment Number is not valid

n INPUT Enrollment Number

END WHILE

1 CREATE new student record with Enrollment Number

1 INPUT Name

n WHILE Name is not valid

n INPUT Name

END WHILE

1 INPUT Father's Name

n WHILE Father's Name is not valid

n INPUT Father's Name

END WHILE

1 INPUT Semester of Admission (July/January)

n WHILE Semester is not valid

1 INPUT Semester of Admission

END WHILE

1 INPUT Year of Admission

n WHILE Year is not valid

n INPUT Year of Admission

END WHILE

1 INPUT Date of Birth (Day, Month, Year)

n WHILE Date of Birth is not valid

n INPUT Date of Birth

1 END WHILE

1 INPUT Subject Details:

1 Semester

1 Year

1 Subject Code

1 Subject Title

1 Max Marks for Sessional

1 Max Marks for Theory

1 Marks Awarded in Sessional

1 Marks Awarded in Theory

1 IF start is null THEN

1 SET start to the new record

1 ELSE

1 FIND the last record

1 SET the last record's next to the new record

1 SET new record's prev to the last record

END IF

END FUNCTION

* + - 1. **Time Complexity by Step Growth and Space Complexity:**

Time Complexity=26+11n ≈O(n)

Space Complexity=S(1)

* + 1. **Delete a scheme/subject from Scheme Database**
       1. **Pseudocode:**

**Steps:** FUNCTION deleteRecord(start, Enrollment Number)

n {CALL searchRecord with start and Enrollment Number

1 IF record is found THEN

1 IF record's prev is not null THEN

1 SET record's prev next to record's next

1 ELSE

1 SET start to record's next

END IF

1 IF record's next is not null THEN

1 SET record's next prev to record's prev

END IF

1 DELETE the record

1 PRINT "Record deleted successfully!"

1 ELSE

1 PRINT "Record not found!"

END IF}

END FUNCTION

* + - 1. **Time Complexity by Step Growth and Space Complexity:**

Time Complexity=11+n ≈O(n)

Space Complexity=S(1)

* + 1. **Update a scheme/subject in Database:**
       1. **Psuedocode:**

STEPS FUNCTION updateRecord(start, Enrollment Number)

n {CALL searchRecord with start and Enrollment Number

1 IF record is found THEN

1 {PRINT "Enter new details for Enrollment No."}

1 INPUT Name

N WHILE Name is not valid

N INPUT Name

END WHILE

1 INPUT Father's Name

N WHILE Father's Name is not valid

N INPUT Father's Name

END WHILE

1 INPUT Semester of Admission (July/January)

N WHILE Semester is not valid

N INPUT Semester of Admission

END WHILE

1 INPUT Year of Admission

N WHILE Year is not valid

N INPUT Year of Admission

END WHILE

1 INPUT Date of Birth (Day, Month, Year)

N WHILE Date of Birth is not valid

N INPUT Date of Birth

END WHILE

INPUT Subject Details:

1 Semester

1 Year

1 Subject Code

1 Subject Title

1 Max Marks for Sessional

1 Max Marks for Theory

1 Marks Awarded in Sessional

1 Marks Awarded in Theory

1 PRINT "Record updated successfully!"

1 ELSE

1 PRINT "Record not found!"

1 END IF}

END FUNCTION

* + - 1. **Time Complexity by Step Growth and Space Complexity:**

Time Complexity=19+11n ≈O(n)

Space Complexity=S(1)

* + 1. **Print report card of a students**
       1. **Psuedocode:**

STEPS FUNCTION printReportCard(SearchStudent(start,id))

1 IF student is null THEN

1 PRINT "Record not found!"

1 RETURN

END IF

1 PRINT "Report Card for student name"

1 PRINT "Enrollment No."

1 PRINT "Father's Name"

1 PRINT "Date of Birth"

1 PRINT "Semester of Admission"

1 PRINT "Year of Admission"

1 PRINT "Subject Details:"

1 PRINT "Semester, Year"

1 PRINT "Subject Code, Title"

1 PRINT "Max Sessional Marks, Awarded"

1 PRINT "Max Theory Marks, Awarded"

END FUNCTION

* + - 1. **Time Complexity by Step Growth and Space Complexity:**

Time Complexity=15 ≈O(1)

Space Complexity=S(1)

* + 1. **Print report of results complete for a semester:**
       1. **Psuedocode:**

STEPS FUNCTION printSemesterResults(start, Semester, Year)

1 PRINT "Results for Semester Semester, Year Year"

1 SET ptr to start

n WHILE ptr is not null

1 IF ptr's Subject Semester and Year match the given Semester and Year T

1 {PRINT "Enrollment No., Name"

1 PRINT "Subject Code, Title"

1 PRINT "Max Sessional Marks, Awarded"

1 PRINT "Max Theory Marks, Awarded"

1 PRINT "-----------------------------"}

END IF

n MOVE to the next record

END WHILE

END FUNCTION

* + - 1. **Time Complexity by Step Growth and Space Complexity:**

Time Complexity=8+n ≈O(n)

Space Complexity=S(1)

* + 1. **Provide the provision to sort the list on any desired field e.g., Name.**
       1. **Psuedocode:**

STEPS FUNCTION sortRecords(start)

1 IF start is null THEN

1 RETURN

END IF

1 SET swapped to true

n WHILE swapped is true

1 SET swapped to false

1 SET ptr to start

n WHILE ptr's next is not null

1 IF ptr's name is greater than ptr's next's name THEN

n SWAP names and other relevant fields of ptr and ptr's next

1 SET swapped to true

END IF

1 MOVE to the next record

END WHILE

END WHILE

1 PRINT "Records sorted successfully!"

END FUNCTION

* + - 1. **Time Complexity by Step Growth and Space Complexity:**

Time Complexity=9+2n ≈O(n)

Space Complexity=S(1)

**CODE IMPLEMENTATION**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

bool YearValidity(int year)

{

    if(year > 2024 || year < 2002)

        return false;

    else

        return true;

}

bool DATEValidity(int day, int month, int year) {

    if (year > 2024 || day < 1 || day > 31 || month < 1 || month > 12) {

        return false;

    }

    if ((month == 4 || month == 6 || month == 9 || month == 11) && day > 30) {

        return false;

    }

    if (month == 2) {

        bool leapyear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

        if (day > (leapyear ? 29 : 28)) {

            return false;

        }

    }

    return true;

}

struct subject {

    int sem, year;

    string SubCode;

    string SubTitle;

    int MaxSessional, MaxTheory;

    int SMarks, TMarks;

};

struct dateOfBirth {

    int day, year, month;

};

struct student {

    int En\_no, AdYear;

    string AdSem;

    string name;

    string father\_name;

    dateOfBirth DOB;

    subject SUB;

    student\* prev;

    student\* next;

};

typedef student S;

bool EnrollmentValidity(S\*& start, int id) {

    if (id < 100000 || id > 999999) {

        cout << "Enrollment No. is Out of Bounds!\n";

        return false;

    }

    S\* ptr = start;

    while (ptr != nullptr) {

        if (ptr->En\_no == id) {

            cout << "Enrollment No. already exists. Insertion failed.\n";

            return false;

        }

        ptr = ptr->next;

    }

    return true;

}

bool STRValidity(string str) {

    for (char c : str) {

        if (!isalpha(c) && c != '.' && c != ' ') return false;

    }

    return true;

}

S\* searchRecord(S\* start, int enrollment) {

    S\* ptr = start;

    while (ptr != nullptr) {

        if (ptr->En\_no == enrollment)

            return ptr;

        ptr = ptr->next;

    }

    return nullptr;

}

// Function Prototypes

void addRecord(S\*& start) {

    int Enroll;

    cout << "Adding a New Record... " << endl;

    cout << "\nEnter Enrollment No.: ";

    do

    {

        cin >> Enroll;

    } while (!EnrollmentValidity(start, Enroll));

    S\* ptr = new S;

    ptr->En\_no = Enroll;

    ptr->next = nullptr;

    ptr->prev = nullptr;

    cin.ignore();

    do

    {

        cout << "Enter Name: "; getline(cin, ptr->name);

    } while (!STRValidity(ptr->name));

    do

    {

        cout << "Enter Father's Name: "; getline(cin, ptr->father\_name);

    } while (!STRValidity(ptr->father\_name));

    do

    {

        cout << "Enter Semester and Year of Admission: \n\tSem of Admission(July/January):"; getline(cin, ptr->AdSem);

    } while (ptr->AdSem != "July" && ptr->AdSem != "January");

    do

    {

        cout << "\n\tyear of admission:  "; cin >> ptr->AdYear;

    } while (!YearValidity(ptr->AdYear));

    while (true)

    {

        cout << "Enter Date of Birth (DD MM YYYY): ";

        cout << "\n\tDay: "; cin >> ptr->DOB.day;

        cout << "\n\tMonth: "; cin >> ptr->DOB.month;

        cout << "\n\tYear: "; cin >> ptr->DOB.year;

        if (DATEValidity(ptr->DOB.day, ptr->DOB.month, ptr->DOB.year))

            break;

        else

            cout << "try again." << endl;

    }

    cin.ignore();

    cout << "\nEnter Subject Details: ";

    cout << "\n\tSemester of Subject: "; cin >> ptr->SUB.sem;

    cout << "\n\t Year of Subject: "; cin >> ptr->SUB.year;

    cin.ignore();

    cout << "\n\tSubject Code: "; getline(cin, ptr->SUB.SubCode);

    cout << "\n\tSubject Title: "; getline(cin, ptr->SUB.SubTitle);

    cout << "\n\t Max Marks for Sessional: "; cin >> ptr->SUB.MaxSessional;

    cout << "\n\t Max Marks for Theory: "; cin >> ptr->SUB.MaxTheory;

    cout << "\n\t Marks Awarded in Sessional: "; cin >> ptr->SUB.SMarks;

    cout << "\n\t Marks Awarded in Theory: "; cin >> ptr->SUB.TMarks;

    if (start == nullptr) {

        start = ptr;

    } else {

        S\* prev = start;

        while (prev->next != nullptr) {

            prev = prev->next;

        }

        prev->next = ptr;

        ptr->prev = prev;

    }

}

void deleteRecord(S\*& start, int enrollment) {

    S\* ptr = searchRecord(start, enrollment);

    if (ptr == nullptr) {

        cout << "Record not found!" << endl;

        return;

    }

    if (ptr->prev != nullptr)

        ptr->prev->next = ptr->next;

    else

        start = ptr->next;

    if (ptr->next != nullptr)

        ptr->next->prev = ptr->prev;

    delete ptr;

    cout << "Record deleted successfully!" << endl;

}

void updateRecord(S\* start, int enrollment) {

    S\* ptr = searchRecord(start, enrollment);

    if (ptr == nullptr) {

        cout << "Record not found!" << endl;

        return;

    }

    cin.ignore();

    do

    {

        cout << "Enter Name: "; getline(cin, ptr->name);

    } while (!STRValidity(ptr->name));

    do

    {

        cout << "Enter Father's Name: "; getline(cin, ptr->father\_name);

    } while (!STRValidity(ptr->father\_name));

    do

    {

        cout << "Enter Semester and Year of Admission: \n\tSem of Admission(July/January):"; getline(cin, ptr->AdSem);

    } while (ptr->AdSem != "July" && ptr->AdSem != "January");

    do

    {

        cout << "\n\tyear of admission:  "; cin >> ptr->AdYear;

    } while (!YearValidity(ptr->AdYear));

    while (true)

    {

        cout << "Enter Date of Birth (DD MM YYYY): ";

        cout << "\n\tDay: "; cin >> ptr->DOB.day;

        cout << "\n\tMonth: "; cin >> ptr->DOB.month;

        cout << "\n\tYear: "; cin >> ptr->DOB.year;

        if (DATEValidity(ptr->DOB.day, ptr->DOB.month, ptr->DOB.year))

            break;

        else

            cout << "try again." << endl;

    }

    cin.ignore();

    cout << "\nEnter Subject Details: ";

    cout << "\n\tSemester of Subject: "; cin >> ptr->SUB.sem;

    cout << "\n\t Year of Subject: "; cin >> ptr->SUB.year;

    cin.ignore();

    cout << "\n\tSubject Code: "; getline(cin, ptr->SUB.SubCode);

    cout << "\n\tSubject Title: "; getline(cin, ptr->SUB.SubTitle);

    cout << "\n\t Max Marks for Sessional: "; cin >> ptr->SUB.MaxSessional;

    cout << "\n\t Max Marks for Theory: "; cin >> ptr->SUB.MaxTheory;

    cout << "\n\t Marks Awarded in Sessional: "; cin >> ptr->SUB.SMarks;

    cout << "\n\t Marks Awarded in Theory: "; cin >> ptr->SUB.TMarks;

    cout << "Record updated successfully!" << endl;

}

void printReportCard(S\* student) {

    if (student == nullptr) {

        cout << "Record not found!" << endl;

        return;

    }

    cout << "Report Card for " << student->name << endl;

    cout << "Enrollment No.: " << student->En\_no << endl;

    cout << "Father's Name: " << student->father\_name << endl;

    cout << "Date of Birth: " << student->DOB.day << "-" << student->DOB.month << "-" << student->DOB.year << endl;

    cout << "Semester of Admission: " << student->AdSem << endl;

    cout << "Year of Admission: " << student->AdYear << endl;

    cout << "Subject Details:" << endl;

    cout << "Semester: " << student->SUB.sem << ", Year: " << student->SUB.year << endl;

    cout << "Subject Code: " << student->SUB.SubCode << ", Title: " << student->SUB.SubTitle << endl;

    cout << "Max Sessional Marks: " << student->SUB.MaxSessional << ", Awarded: " << student->SUB.SMarks << endl;

    cout << "Max Theory Marks: " << student->SUB.MaxTheory << ", Awarded: " << student->SUB.TMarks << endl;

}

void printSemesterResults(S\* start, int semester, int year) {

    S\* ptr = start;

    cout << "Results for Semester " << semester << ", Year " << year << endl;

    while (ptr != nullptr) {

        if (ptr->SUB.sem == semester && ptr->SUB.year == year) {

            cout << "Enrollment No.: " << ptr->En\_no << ", Name: " << ptr->name << endl;

            cout << "Subject Code: " << ptr->SUB.SubCode << ", Title: " << ptr->SUB.SubTitle << endl;

            cout << "Max Sessional Marks: " << ptr->SUB.MaxSessional << ", Awarded: " << ptr->SUB.SMarks << endl;

            cout << "Max Theory Marks: " << ptr->SUB.MaxTheory << ", Awarded: " << ptr->SUB.TMarks << endl;

            cout << "-----------------------------" << endl;

        }

        ptr = ptr->next;

    }

}

void sortRecords(S\*& start) {

    if (start == nullptr) return;

    bool swapped;

    do {

        swapped = false;

        S\* ptr = start;

        while (ptr->next != nullptr) {

            if (ptr->name > ptr->next->name) {

                swap(ptr->name, ptr->next->name);

                swap(ptr->En\_no, ptr->next->En\_no);

                swap(ptr->father\_name, ptr->next->father\_name);

                swap(ptr->DOB, ptr->next->DOB);

                swap(ptr->AdSem, ptr->next->AdSem);

                swap(ptr->AdYear, ptr->next->AdYear);

                swap(ptr->SUB, ptr->next->SUB);

                swapped = true;

            }

            ptr = ptr->next;

        }

    } while (swapped);

    cout << "Records sorted successfully!" << endl;

}

void saveToFile(S\* start) {

    ofstream file("students.txt");

    S\* ptr = start;

    while (ptr != nullptr) {

        file << ptr->En\_no << endl;

        file << ptr->name << endl;

        file << ptr->father\_name << endl;

        file << ptr->AdSem << endl;

        file << ptr->AdYear << endl;

        file << ptr->DOB.day << " " << ptr->DOB.month << " " << ptr->DOB.year << endl;

        file << ptr->SUB.sem << " " << ptr->SUB.year << endl;

        file << ptr->SUB.SubCode << endl;

        file << ptr->SUB.SubTitle << endl;

        file << ptr->SUB.MaxSessional << endl;

        file << ptr->SUB.MaxTheory << endl;

        file << ptr->SUB.SMarks << endl;

        file << ptr->SUB.TMarks << endl;

        ptr = ptr->next;

    }

    file.close();

    cout << "Data saved to text file successfully!" << endl;

}

void loadFromFile(S\*& start) {

    ifstream file("students.txt");

    if (!file) {

        cout << "Error opening file!" << endl;

        return;

    }

    while (!file.eof()) {

        S\* ptr = new S;

        file >> ptr->En\_no;

        file.ignore(); // ignore newline character after En\_no

        getline(file, ptr->name);

        getline(file, ptr->father\_name);

        getline(file, ptr->AdSem);

        file >> ptr->AdYear;

        file.ignore(); // ignore newline character after AdYear

        file >> ptr->DOB.day >> ptr->DOB.month >> ptr->DOB.year;

        file >> ptr->SUB.sem >> ptr->SUB.year;

        file.ignore(); // ignore newline character after SUB.year

        getline(file, ptr->SUB.SubCode);

        getline(file, ptr->SUB.SubTitle);

        file >> ptr->SUB.MaxSessional;

        file >> ptr->SUB.MaxTheory;

        file >> ptr->SUB.SMarks;

        file >> ptr->SUB.TMarks;

        file.ignore(); // ignore newline character after TMarks

        ptr->next = nullptr;

        ptr->prev = nullptr;

        if (start == nullptr) {

            start = ptr;

        } else {

            S\* last = start;

            while (last->next != nullptr) {

                last = last->next;

            }

            last->next = ptr;

            ptr->prev = last;

        }

    }

    file.close();

    cout << "Data loaded from file successfully!" << endl;

}

int main() {

    S\* start = nullptr;

    loadFromFile(start);

    int choice, enrollment, semester, year;

    while (true) {

        cout << "\n1. Add Record\n2. Delete Record\n3. Update Record\n4. Print Report Card\n5. Print Semester Results\n6. Sort Records\n7. Save to File\n8. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                addRecord(start);

                break;

            case 2:

                cout << "Enter Enrollment Number to Delete: ";

                cin >> enrollment;

                deleteRecord(start, enrollment);

                break;

            case 3:

                cout << "Enter Enrollment Number to Update: ";

                cin >> enrollment;

                updateRecord(start, enrollment);

                break;

            case 4:

                cout << "Enter Enrollment Number to Print Report Card: ";

                cin >> enrollment;

                printReportCard(searchRecord(start, enrollment));

                break;

            case 5:

                cout << "Enter Semester and Year to Print Results: ";

                cin >> semester >> year;

                printSemesterResults(start, semester, year);

                break;

            case 6:

                sortRecords(start);

                break;

            case 7:

                saveToFile(start);

                break;

            case 8:

                return 0;

            default:

                cout << "Invalid choice, please try again!" << endl;

        }

    }

    return 0;

}

**CODE RESULTS**

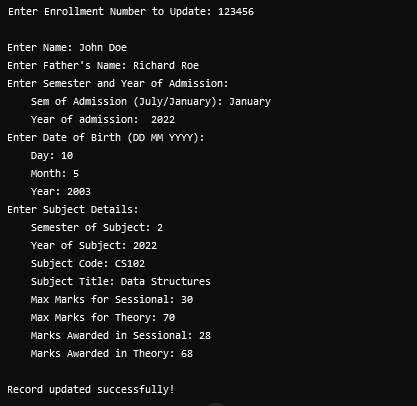
1. **Add a scheme in Database:**

****

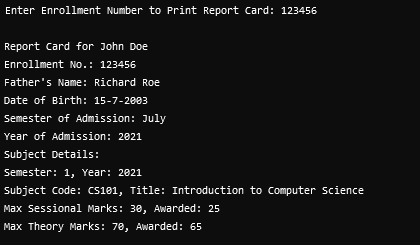
1. **Delete a scheme/subject from Scheme Database**

****

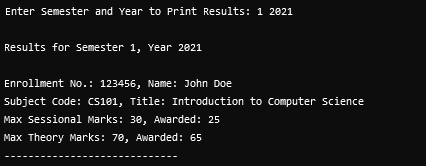
1. **Update a scheme/subject in Database:**

****

1. **Print report card of a students**

****

1. **Print report of results complete for a semester:**

****

1. **Provide the provision to sort the list on any desired field e.g., Name.**

