Course Code	**:**	**BCSL-032**
Course Title	**:**	**C++Programming Lab**
Assignment Number	**:**	**BCA(III)/L-032/Assignment/2024-25**
Maximum Marks	**:**	**50**
Weightage	**:**	**25%**
Last date of Submission	**:**	**31st October,2024(for July session)** **30th April,2025(for January session)**

This assignment has two questions. Answer both the questions. These questions carry 40 marks. Rest 10 marks are for viva-voce. Write C++ program and take its output as part of solution. Please go through the guidelines regarding the assignments given in the Programme Guide for the format of presentation.

Q1. Write a program in C++ for addition of two sparse Polynomials using Pointers. **(20 Marks)**

Q2. Write a program in C++ to generate ranks for the candidates based on the marks secured by them in an entrance examination. Make necessary assumptions. **(10 Marks)**

Q3. Write a program in C++ to create a book of 10 input pages. Make necessary assumptions. **(10 Marks)**

Q1. Write a program in C++ for addition of two sparse Polynomials using Pointers

Ans

Here's a C++ program to add two sparse polynomials using pointers:

Code C++

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

// Node structure to represent a term in the polynomial
struct Node {
    int coeff;  // Coefficient of the term
    int power;  // Power of the term
    Node* next;
};

// Function to create a new node
Node* createNode(int coeff, int power) {
    Node* newNode = new Node;
    newNode->coeff = coeff;
    newNode->power = power;
    newNode->next = nullptr;
    return newNode;
}

// Function to insert a new term in the polynomial
void insertTerm(Node*& poly, int coeff, int power) {
    Node* newNode = createNode(coeff, power);

    // If the list is empty or power is greater than the first node's power
    if (!poly || poly->power < power) {
```

```
        newNode->next = poly;
        poly = newNode;
    } else {
        Node* temp = poly;
        while (temp->next && temp->next->power >= power) {
            temp = temp->next;
        }

        // If the power already exists, add the coefficients
        if (temp->power == power) {
            temp->coeff += coeff;
        } else {
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
}

// Function to add two polynomials
Node* addPolynomials(Node* poly1, Node* poly2) {
    Node* result = nullptr;

    while (poly1 && poly2) {
        if (poly1->power > poly2->power) {
            insertTerm(result, poly1->coeff, poly1->power);
            poly1 = poly1->next;
        } else if (poly2->power > poly1->power) {
            insertTerm(result, poly2->coeff, poly2->power);
            poly2 = poly2->next;
```

```
        } else {
            insertTerm(result, poly1->coeff + poly2->coeff, poly1->power);
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }

    // If there are remaining terms in poly1
    while (poly1) {
        insertTerm(result, poly1->coeff, poly1->power);
        poly1 = poly1->next;
    }

    // If there are remaining terms in poly2
    while (poly2) {
        insertTerm(result, poly2->coeff, poly2->power);
        poly2 = poly2->next;
    }

    return result;
}

// Function to print a polynomial
void printPolynomial(Node* poly) {
    while (poly) {
        cout << poly->coeff << "x^" << poly->power;
        poly = poly->next;
        if (poly) {
            cout << " + ";
```

```
        }
    }
    cout << endl;
}

// Main function
int main() {
    Node* poly1 = nullptr;
    Node* poly2 = nullptr;

    // First polynomial: 5x^3 + 4x^2 + 2
    insertTerm(poly1, 5, 3);
    insertTerm(poly1, 4, 2);
    insertTerm(poly1, 2, 0);

    // Second polynomial: 3x^3 + 4x + 6
    insertTerm(poly2, 3, 3);
    insertTerm(poly2, 4, 1);
    insertTerm(poly2, 6, 0);

    cout << "First Polynomial: ";
    printPolynomial(poly1);

    cout << "Second Polynomial: ";
    printPolynomial(poly2);

    Node* result = addPolynomials(poly1, poly2);

    cout << "Resultant Polynomial: ";
```

```
    printPolynomial(result);


    return 0;
}
```

Explanation:

1. **Node Structure**: Represents each term in the polynomial, containing the coefficient, power, and a pointer to the next node.
2. **createNode Function**: Allocates memory for a new node and initializes it with the given coefficient and power.
3. **insertTerm Function**: Inserts a term in the correct position in the polynomial. If the term with the same power exists, it adds the coefficients.
4. **addPolynomials Function**: Adds two polynomials by iterating through both lists, comparing the powers, and adding the coefficients of like terms.
5. **printPolynomial Function**: Prints the polynomial in a readable format.
6. **Main Function**: Demonstrates the addition of two sample polynomials.

Sample Output:

First Polynomial: 5x^3 + 4x^2 + 2x^0

Second Polynomial: 3x^3 + 4x^1 + 6x^0

Resultant Polynomial: 8x^3 + 4x^2 + 4x^1 + 8x^0

This code will work correctly for adding sparse polynomials using pointers.

Q2. Write a program in C++ to generate ranks for the candidates based on the marks secured by them in an entrance examination. Make necessary assumptions.

Ans

Here's a C++ program to generate ranks for candidates based on the marks they secured in an entrance examination:

Code C++

```
#include <iostream>

#include <vector>

#include <algorithm>
```

```
using namespace std;

// Structure to represent a candidate
struct Candidate {
    int id;          // Candidate ID
    string name;     // Candidate Name
    int marks;       // Marks secured by the candidate
    int rank;        // Rank of the candidate
};

// Comparison function to sort candidates by marks in descending order
bool compareByMarks(const Candidate& a, const Candidate& b) {
    return a.marks > b.marks;
}

// Function to assign ranks to candidates
void assignRanks(vector<Candidate>& candidates) {
    // Sort the candidates by marks in descending order
    sort(candidates.begin(), candidates.end(), compareByMarks);

    // Assign ranks based on sorted order
    for (size_t i = 0; i < candidates.size(); i++) {
        // Check for candidates with the same marks (tie case)
        if (i > 0 && candidates[i].marks == candidates[i - 1].marks) {
            candidates[i].rank = candidates[i - 1].rank;  // Same rank as the previous candidate
        } else {
            candidates[i].rank = i + 1;  // Assign rank based on position
        }
```

```
    }
}

// Function to print the list of candidates with their ranks
void printCandidates(const vector<Candidate>& candidates) {
    cout << "Rank\tID\tName\tMarks\n";
    for (const auto& candidate : candidates) {
        cout << candidate.rank << "\t" << candidate.id << "\t" << candidate.name << "\t" << candidate.marks << endl;
    }
}

// Main function
int main() {
    int numCandidates;

    cout << "Enter the number of candidates: ";
    cin >> numCandidates;

    vector<Candidate> candidates(numCandidates);

    // Input candidate details
    for (int i = 0; i < numCandidates; i++) {
        cout << "Enter details for candidate " << i + 1 << ":\n";
        cout << "ID: ";
        cin >> candidates[i].id;
        cout << "Name: ";
        cin >> candidates[i].name;
        cout << "Marks: ";
```

```
        cin >> candidates[i].marks;
    }

    // Assign ranks to the candidates
    assignRanks(candidates);

    // Print the ranked list of candidates
    cout << "\nRanked List of Candidates:\n";
    printCandidates(candidates);

    return 0;
}
```

Explanation:

1. **Candidate Structure**:
    - Contains id (candidate ID), name (candidate name), marks (marks secured), and rank (rank of the candidate).
2. **compareByMarks Function**:
    - Used to sort candidates by marks in descending order. It's a comparator function that returns true if the marks of the first candidate are greater than the second.
3. **assignRanks Function**:
    - First, it sorts the list of candidates based on their marks using the compareByMarks function.
    - Then, it assigns ranks to the candidates. If two or more candidates have the same marks, they are given the same rank.
4. **printCandidates Function**:
    - Prints the list of candidates along with their rank, ID, name, and marks in a tabular format.
5. **Main Function**:
    - Prompts the user to input the number of candidates and their details (ID, name, marks).

- Calls the assignRanks function to determine the ranks.
- Finally, it prints the ranked list of candidates.

Sample Output:

Enter the number of candidates: 5

Enter details for candidate 1:

ID: 101

Name: Alice

Marks: 95

Enter details for candidate 2:

ID: 102

Name: Bob

Marks: 88

Enter details for candidate 3:

ID: 103

Name: Charlie

Marks: 95

Enter details for candidate 4:

ID: 104

Name: David

Marks: 82

Enter details for candidate 5:

ID: 105

Name: Eva

Marks: 88

Ranked List of Candidates:

Rank ID Name Marks

1 101 Alice 95

1 103 Charlie 95

3 102 Bob 88

3 105 Eva 88

5 104 David 82

In this example:

- Alice and Charlie both have the highest marks, so they share the 1st rank.
- Bob and Eva have the next highest marks, so they share the 3rd rank.
- David has the lowest marks, so he is ranked 5th.

This program assumes that the number of candidates and their details (ID, name, and marks) are input by the user. The program handles ties by assigning the same rank to candidates with equal marks.

Q3. Write a program in C++ to create a book of 10 input pages. Make necessary assumptions.

Ans

Here's a C++ program that simulates creating a book of 10 pages. Each page will have a page number and some content entered by the user.

Code C++

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

class Page {
public:
    int pageNumber;
    string content;

    void setPage(int num, string cont) {
        pageNumber = num;
        content = cont;
    }

    void displayPage() {
```

```
        cout << "Page " << pageNumber << ": " << content << endl;
    }
};

class Book {
public:
    Page pages[10];  // Array of 10 pages

    void createBook() {
        for (int i = 0; i < 10; ++i) {
            string content;
            cout << "Enter content for page " << i + 1 << ": ";
            getline(cin, content);  // Get page content from the user
            pages[i].setPage(i + 1, content);
        }
    }

    void displayBook() {
        cout << "\nBook Contents:" << endl;
        for (int i = 0; i < 10; ++i) {
            pages[i].displayPage();
        }
    }
};

int main() {
    Book myBook;
    myBook.createBook();   // Create the book with 10 pages
    myBook.displayBook();  // Display the book content
```

```
    return 0;
}
```

Explanation:

1. **Page Class:**
    - Contains the page number and content.
    - Methods to set the page details and display them.
2. **Book Class:**
    - Holds an array of 10 Page objects.
    - createBook() method asks the user for content for each of the 10 pages.
    - displayBook() method shows the content of each page.
3. **Main Function:**
    - Creates an instance of the Book class.
    - Calls methods to create and display the book's content.

Assumptions:

- The book has exactly 10 pages.
- Each page has a number and content entered by the user.
- The book is displayed with the page number and corresponding content.