DS Practical File

August – 2025

Natya Vidhan Biswas

25771, B.Sc. Hons. CS.

1. Write a menu driven C++ Program to perform the following operations on an integer:

- Sum of digits
- Reversing the integer
- Factorial
- Printing Fibonacci series upto integer n
- Printing prime number upto integer n

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

int sum(int n) {
    int sum = 0;
    while (n) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}

int reverse(int n) {
    int rev = 0;
    while (n) {
        rev = rev * 10 + n % 10;
        n /= 10;
    }
    return rev;
}

int factorial(int n) {
    int fact = 1;
    for (int i = 2; i <= n; i++)
        fact *= i;
    return fact;
}

void fibonacci(int n) {
    int a = 0, b = 1, c;
    cout << "Fibonacci series: " << a << " " << b << " ";
    for (int i = 2; i < n; i++) {
        c = a + b;
        cout << c << " ";
        a = b;
        b = c;
    }
    cout << endl;
}
```

```
bool is_prime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return false;
    }
    return true;
}

void print_primes(int n) {
    cout << "Primes up to " << n << ": ";
    for (int i = 2; i <= n; i++) {
        if (is_prime(i))
            cout << i << " ";
    }
    cout << endl;
}

int main() {
    int choice, n;
    while (true) {
        cout << "\n1. Sum of digits\n2. Reverse integer\n3.
Factorial\n4. Fibonacci series\n5. Prime numbers\n6. Exit\nEnter
choice: ";
        cin >> choice;
        if (choice == 6) break;
        cout << "Enter an integer: ";
        cin >> n;
        if (choice == 1) {
            cout << "Sum of digits = " << sum(n) << endl;
        } else if (choice == 2) {
            cout << "Reversed number = " << reverse(n) << endl;
        } else if (choice == 3) {
            cout << "Factorial = " << factorial(n) << endl;
        } else if (choice == 4) {
            fibonacci(n);
        } else if (choice == 5) {
            print_primes(n);
        } else {
            cout << "Invalid choice\n";
        }
    }
    return 0;
}
```

2. Write a menu driven C++ Program to perform the following operations on an array of integers: Use dynamically allocated 1 D array

- Sum of elements of the array
- Average of elements of the array
- Minimum and Maximum element of the array
- Sorting the array using Selection sort
- Sorting the array using Insertion sort
- Searching an element in the array
- Merging two arrays
- Concatenating two arrays

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


int sum(int *arr, int n) {
    int total = 0;
    for(int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

double average(int *arr, int n) {
    return (double)sum(arr, n) / n;
}

void min_max(int *arr, int n, int &min, int &max) {
    min = max = arr[0];
    for(int i = 1; i < n; i++) {
        if(arr[i] < min) min = arr[i];
        if(arr[i] > max) max = arr[i];
    }
}

void selection_sort(int *arr, int n) {
    for(int i = 0; i < n-1; i++) {
        int minIdx = i;
        for(int j = i+1; j < n; j++)
            if(arr[j] < arr[minIdx])
                minIdx = j;
        swap(arr[i], arr[minIdx]);
    }
    cout << "Array sorted using Selection Sort\n";
}



void insertion_sort(int *arr, int n) {
```

```
    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while(j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
    cout << "Array sorted using Insertion Sort\n";
}

void search(int *arr, int n, int key) {
    for(int i = 0; i < n; i++) {
        if(arr[i] == key) {
            cout << "Element found at index " << i << endl;
            return;
        }
    }
    cout << "Element not found\n";
}

int* merge_arrays(int *arr1, int n1, int *arr2, int n2) {
    int *merged = new int[n1 + n2];
    for(int i = 0; i < n1; i++)
        merged[i] = arr1[i];
    for(int i = 0; i < n2; i++)
        merged[n1 + i] = arr2[i];
    return merged;
}

int* concatenate_arrays(int *arr1, int n1, int *arr2, int n2) {
    return merge_arrays(arr1, n1, arr2, n2);
}

void print_array(int *arr, int n) {
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}


int main() {
    int choice, n;
    cout << "Enter size of array: ";
    cin >> n;
    int *arr = new int[n];
    cout << "Enter elements:\n";
    for(int i = 0; i < n; i++)
        cin >> arr[i];

    while (true) {
        cout << "\n1. Sum of elements\n2. Average of elements\n3. Minimum
and Maximum\n4. Sort using Selection Sort\n";
```

```
        cout << "5. Sort using Insertion Sort\n6. Search an element\n7.
Merge with another array\n";
        cout << "8. Concatenate with another array\n9. Display array\n10.
Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        if (choice == 10)
            break;

        if (choice == 1) {
            cout << "Sum = " << sum(arr, n) << endl;
        } else if (choice == 2) {
            cout << "Average = " << average(arr, n) << endl;
        } else if (choice == 3) {
            int minVal, maxVal;
            min_max(arr, n, minVal, maxVal);
            cout << "Minimum = " << minVal << ", Maximum = " << maxVal <<
endl;
        } else if (choice == 4) {
            selection_sort(arr, n);
            print_array(arr, n);
        } else if (choice == 5) {
            insertion_sort(arr, n);
            print_array(arr, n);
        } else if (choice == 6) {
            int key;
            cout << "Enter element to search: ";
            cin >> key;
            search(arr, n, key);
        } else if (choice == 7) {
            int n2;
            cout << "Enter size of second array: ";
            cin >> n2;
            int *arr2 = new int[n2];
            cout << "Enter elements of second array:\n";
            for(int i = 0; i < n2; i++)
                cin >> arr2[i];
            int *merged = merge_arrays(arr, n, arr2, n2);
            cout << "Merged array:\n";
            print_array(merged, n + n2);
            delete[] arr2;
            delete[] merged;
        } else if (choice == 8) {
            int n2;
            cout << "Enter size of second array: ";
            cin >> n2;
            int *arr2 = new int[n2];
            cout << "Enter elements of second array:\n";
            for(int i = 0; i < n2; i++)
                cin >> arr2[i];
            int *concat = concatenate_arrays(arr, n, arr2, n2);
            cout << "Concatenated array:\n";
            print_array(concat, n + n2);
```

```
            delete[] arr2;
            delete[] concat;
        } else if (choice == 9) {
            cout << "Array elements:\n";
            print_array(arr, n);
        } else {
            cout << "Invalid choice\n";
        }
    }

    delete[] arr;
    return 0;
}
```

3. Write a menu driven C++ Program to perform the following matrix operations on a 2d array of integers: Use a dynamically allocated 2 D array

- Addition of two matrices
- Multiplication of 2 matrices
- Transpose of a matrix

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

int** createMatrix(int rows, int cols) {
    int **mat = new int*[rows];
    for(int i = 0; i < rows; i++)
        mat[i] = new int[cols];
    return mat;
}

void inputMatrix(int **mat, int rows, int cols) {
    cout << "Enter matrix elements:\n";
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            cin >> mat[i][j];
}

void printMatrix(int **mat, int rows, int cols) {
    cout << "Matrix:\n";
    for(int i = 0; i < rows; i++) {
        for(int j = 0; j < cols; j++)
            cout << mat[i][j] << " ";
        cout << endl;
    }
}

void addMatrices(int **mat1, int **mat2, int rows, int cols) {
    int **result = createMatrix(rows, cols);
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            result[i][j] = mat1[i][j] + mat2[i][j];
    cout << "Addition Result:\n";
    printMatrix(result, rows, cols);
    for(int i = 0; i < rows; i++)
        delete[] result[i];
    delete[] result;
}


void multiplyMatrices(int **mat1, int r1, int c1, int **mat2, int r2, int
c2) {
    if(c1 != r2) {
        cout << "Multiplication not possible due to dimension
mismatch!\n";
        return;
    }
```

```
    int **result = createMatrix(r1, c2);
    for(int i = 0; i < r1; i++) {
        for(int j = 0; j < c2; j++) {
            result[i][j] = 0;
            for(int k = 0; k < c1; k++)
                result[i][j] += mat1[i][k] * mat2[k][j];
        }
    }
    cout << "Multiplication Result:\n";
    printMatrix(result, r1, c2);
    for(int i = 0; i < r1; i++)
        delete[] result[i];
    delete[] result;
}

void transposeMatrix(int **mat, int rows, int cols) {
    int **result = createMatrix(cols, rows);
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            result[j][i] = mat[i][j];
    cout << "Transpose of the matrix:\n";
    printMatrix(result, cols, rows);
    for(int i = 0; i < cols; i++)
        delete[] result[i];
    delete[] result;
}

void deleteMatrix(int **mat, int rows) {
    for(int i = 0; i < rows; i++)
        delete[] mat[i];
    delete[] mat;
}

int main() {
    int choice;
    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Add two matrices\n";
        cout << "2. Multiply two matrices\n";
        cout << "3. Transpose a matrix\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 4)
            break;

        if (choice == 1) {
            int rows, cols;
            cout << "Enter number of rows and columns: ";
            cin >> rows >> cols;
            int **mat1 = createMatrix(rows, cols);
            int **mat2 = createMatrix(rows, cols);
            cout << "Enter elements of first matrix:\n";
```

```
            inputMatrix(mat1, rows, cols);
            cout << "Enter elements of second matrix:\n";
            inputMatrix(mat2, rows, cols);
            addMatrices(mat1, mat2, rows, cols);
            deleteMatrix(mat1, rows);
            deleteMatrix(mat2, rows);
        } else if (choice == 2) {
            int r1, c1, r2, c2;
            cout << "Enter rows and columns of first matrix: ";
            cin >> r1 >> c1;
            cout << "Enter rows and columns of second matrix: ";
            cin >> r2 >> c2;
            int **mat1 = createMatrix(r1, c1);
            int **mat2 = createMatrix(r2, c2);
            cout << "Enter elements of first matrix:\n";
            inputMatrix(mat1, r1, c1);
            cout << "Enter elements of second matrix:\n";
            inputMatrix(mat2, r2, c2);
            multiplyMatrices(mat1, r1, c1, mat2, r2, c2);
            deleteMatrix(mat1, r1);
            deleteMatrix(mat2, r2);
        } else if (choice == 3) {
            int rows, cols;
            cout << "Enter number of rows and columns: ";
            cin >> rows >> cols;
            int **mat = createMatrix(rows, cols);
            cout << "Enter elements of the matrix:\n";
            inputMatrix(mat, rows, cols);
            transposeMatrix(mat, rows, cols);
            deleteMatrix(mat, rows);
        } else {
            cout << "Invalid choice!\n";
        }
    }
    cout << "Program exited.\n";
    return 0;
}
```

4. Game entries Program

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

class GameEntry
{
    string name;
    int score;

public:
    GameEntry(string n = "", int s = 0)
    {
        name = n;
        score = s;
    }

    string get_name() const { return name; }
    int get_score() const { return score; }
};

class Scores
{
    GameEntry *entries;
    int maxEntries;
    int numEntries;

public:
    Scores(int n = 10)
    {
        maxEntries = n;
        numEntries = 0;
        entries = new GameEntry[maxEntries];
    }

    ~Scores()
    {
        delete[] entries;
    }

    void add_score(const GameEntry &e)
    {
        int s = e.get_score();

        if (numEntries == maxEntries && s <= entries[numEntries -
1].get_score())
        {
            cout << "?? Score not high enough to enter the
leaderboard.\n";
            return;
        }
```

```
    if (numEntries < maxEntries)
        numEntries++;

    int i = numEntries - 2;
    while (i >= 0 && entries[i].get_score() < s)
    {
        entries[i + 1] = entries[i];
        i--;
    }
    entries[i + 1] = e;
}

void delete_by_name(const string &name)
{
    int idx = -1;
    for (int i = 0; i < numEntries; i++)
    {
        if (entries[i].get_name() == name)
        {
            idx = i;
            break;
        }
    }

    if (idx == -1)
    {
        cout << "Player not found!\n";
        return;
    }

    for (int i = idx; i < numEntries - 1; i++)
    {
        entries[i] = entries[i + 1];
    }
    numEntries--;
    cout << "Entry for " << name << " deleted.\n";
}

void delete_last()
{
    if (numEntries == 0)
    {
        cout << "No entries to delete!\n";
        return;
    }
    numEntries--;
    cout << "Last entry deleted.\n";
}
```

```
    void display()
    {
        cout << "\n--- Leaderboard ---\n";
        for (int i = 0; i < numEntries; i++)
        {
            cout << i + 1 << ". " << entries[i].get_name()
                 << " : " << entries[i].get_score() << endl;
        }
        if (numEntries == 0)
            cout << "No entries yet.\n";
    }
};

int main()
{
    int capacity;
    cout << "Enter leaderboard size (max entries): ";
    cin >> capacity;

    Scores obj(capacity);

    int choice;
    while (true)
    {
        cout << "\nGame Entry Menu\n";
        cout << "1. Add new entry\n";
        cout << "2. Display leaderboard\n";
        cout << "3. Delete entry by name\n";
        cout << "4. Delete last entry\n";
        cout << "5. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        if (choice == 1)
        {
            string name;
            int score;
            cout << "Enter player name: ";
            cin >> name;
            cout << "Enter player score: ";
            cin >> score;
            GameEntry e(name, score);
            obj.add_score(e);
        }
        else if (choice == 2)
        {
            obj.display();
        }
        else if (choice == 3)
        {
            string name;
            cout << "Enter player name to delete: ";
            cin >> name;
            obj.delete_by_name(name);
```

```
        }
        else if (choice == 4)
        {
            obj.delete_last();
        }
        else if (choice == 5)
        {
            cout << "Exiting program...\n";
            break;
        }
        else
        {
            cout << "Invalid choice\n";
        }
    }

    return 0;
}
```

5. Write a program to implement singly linked list as an ADT that supports the following operations:

- Insert an element x at the beginning of the singly linked list
- Insert an element x at ith position in the singly linked list
- Insert an element x at the end of the singly linked list
- Remove an element from the beginning of the singly linked list
- Remove an element from ith position in the singly linked list.
- Remove an element from the end of the singly linked list
- Search for an element x in the singly linked list and return its pointer
- Reverse a singly linked list
- Create an ordered linked list

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

struct Node {
    int data;
    Node* next;
};
void insert_at_beginning(Node*& head, int x) {
    Node* newNode = new Node{x, head};
    head = newNode;
}
void insert_at_position(Node*& head, int x, int pos) {
    if (pos <= 1 || head == nullptr) {
        insert_at_beginning(head, x);
        return;
    }
    Node* temp = head;
    for (int i = 1; temp != nullptr && i < pos - 1; i++) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        cout << "Position out of range.\n";
        return;
    }
    Node* newNode = new Node{x, temp->next};
    temp->next = newNode;
}
void insert_at_end(Node*& head, int x) {
    Node* newNode = new Node{x, nullptr};
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
```

```
    while (temp->next) temp = temp->next;
    temp->next = newNode;
}
void remove_beginning(Node*& head) {
    if (!head) {
        cout << "List is empty.\n";
        return;
    }
    Node* temp = head;
    head = head->next;
    delete temp;
}
void remove_at_position(Node*& head, int pos) {
    if (!head) {
        cout << "List is empty.\n";
        return;
    }
    if (pos == 1) {
        remove_beginning(head);
        return;
    }
    Node* temp = head;
    for (int i = 1; temp && i < pos - 1; i++) {
        temp = temp->next;
    }
    if (!temp || !temp->next) {
        cout << "Position out of range.\n";
        return;
    }
    Node* del = temp->next;
    temp->next = temp->next->next;
    delete del;
}
void remove_end(Node*& head) {
    if (!head) {
        cout << "List is empty.\n";
        return;
    }
    if (!head->next) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next->next) temp = temp->next;
    delete temp->next;
    temp->next = nullptr;
}
Node* search(Node* head, int x) {
    Node* temp = head;
    while (temp) {
        if (temp->data == x) return temp;
        temp = temp->next;
    }
```

```
    return nullptr;
}
void reverse_list(Node*& head) {
    Node* prev = nullptr;
    Node* curr = head;
    while (curr) {
        Node* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;
    }
    head = prev;
}
void insert_ordered(Node*& head, int x) {
    Node* newNode = new Node{x, nullptr};
    if (!head || head->data >= x) {
        newNode->next = head;
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next && temp->next->data < x) {
        temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}
Node* merge_lists(Node* head1, Node* head2) {
    Node* result = nullptr;
    Node* temp1 = head1;
    Node* temp2 = head2;
    while (temp1) {
        insert_ordered(result, temp1->data);
        temp1 = temp1->next;
    }
    while (temp2) {
        insert_ordered(result, temp2->data);
        temp2 = temp2->next;
    }
    return result;
}
void print_list(Node* head) {
    cout << "List: ";
    Node* temp = head;
    while (temp) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}
```

```
int main() {
    Node* head = nullptr;
    int choice, x, pos;

    while (true) {
        cout << "\n=== MENU ===\n";
        cout << "1. Insert at beginning\n";
        cout << "2. Insert at ith position\n";
        cout << "3. Insert at end\n";
        cout << "4. Remove from beginning\n";
        cout << "5. Remove from ith position\n";
        cout << "6. Remove from end\n";
        cout << "7. Search element\n";
        cout << "8. Reverse list\n";
        cout << "9. Insert in ordered list\n";
        cout << "10. Merge two lists\n";
        cout << "11. Print list\n";
        cout << "12. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1) {
            cout << "Enter element: ";
            cin >> x;
            insert_at_beginning(head, x);
        } else if (choice == 2) {
            cout << "Enter element and position: ";
            cin >> x >> pos;
            insert_at_position(head, x, pos);
        } else if (choice == 3) {
            cout << "Enter element: ";
            cin >> x;
            insert_at_end(head, x);
        } else if (choice == 4) {
            remove_beginning(head);
        } else if (choice == 5) {
            cout << "Enter position: ";
            cin >> pos;
            remove_at_position(head, pos);
        } else if (choice == 6) {
            remove_end(head);
        } else if (choice == 7) {
            cout << "Enter element to search: ";
            cin >> x;
            if (search(head, x))
                cout << "Element " << x << " found.\n";
            else
                cout << "Element not found.\n";
        } else if (choice == 8) {
            reverse_list(head);
            cout << "List reversed.\n";
        } else if (choice == 9) {
            cout << "Enter element: ";
            cin >> x;
```

```
            insert_ordered(head, x);
        } else if (choice == 10) {
            Node* head2 = nullptr;
            int n, val;
            cout << "Enter number of elements for 2nd list: ";
            cin >> n;
            cout << "Enter elements:\n";
            for (int i = 0; i < n; i++) {
                cin >> val;
                insert_ordered(head2, val);
            }
            Node* merged = merge_lists(head, head2);
            cout << "Merged Ordered List:\n";
            print_list(merged);
        } else if (choice == 11) {
            print_list(head);
        } else if (choice == 12) {
            cout << "Exiting program...\n";
            break;
        } else {
            cout << "Invalid choice! Try again.\n";
        }
    }

    return 0;
}
```