

LAB MANUAL

Data structure and algorithm



Submitted to: Mam Irsha

Submitted by: Haris Awan

REGISTRATION NO: 2023-BS-AI-023

Table of Contents

Lab 1: Introduction to DSA	2
Lab 2: Array.....	4
Lab 3: 2d Array.....	8
Lab 4: Vector.....	13
Lab 5: List	16
Lab 6: Stacks	18
Lab 7: Queues	23
Lab 8: Quiz.....	26
Lab 9: Dequeue	30
Lab 10: Singly link list.....	33
Lab 11: Doubly link list	45
Lab 12: Circular link list	48
Lab 13: Trees.....	52
Lab 14: Binary Search trees.....	56

Lab 1: Introduction to DSA

What is data structure?

- Data structure are fundamental constructs that allow us to organize and store data efficiently, making it easier to perform various operations such as searching, sorting, and manipulating the data.

Data structures in c++:

- Array
- Vector
- Link list
- Stack
- Queue

What are algorithms

- An **algorithm** is a step-by-step procedure or set of rules for solving a specific problem or performing a specific task. It is a well-defined, finite sequence of instructions that takes an input, processes it, and produces an output

Types of data structures

- There are two types of data structures
 1. **Primitive Data Structures:** These are the fundamental building blocks provided by programming languages, used to represent simple data values.
- **Features of Primitive Data Structures:**
 - ❖ Represent single values.
 - ❖ Directly supported by hardware or programming languages.
 - ❖ Simple to use and manipulate.
- 2. **Abstract Data Structures:** Abstract Data Structures are conceptual models that define how data is stored, accessed, and manipulated without specifying the implementation details.

- **Linear:** Which have fixed size like arrays, list, stack and queue

- **Non-Linear:** Which do not have fixed size like trees and graphs

Where are data structures used

- Networking
- Database Systems
- Web Applications
- Machine Learning
- Video Games
- Software development
- Artificial Intelligence and machine learning

LAB 2 : ARRAY

Definition:

An **array** is a **linear data structure** that stores a collection of elements, typically of the same data type, in **contiguous memory locations**. Arrays allow random access to elements using indices and are widely used for their simplicity and efficient access times

Syntax:

```
int numbers[5]; // Array of 5 integers
```

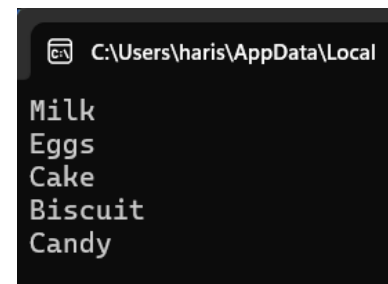
```
int numbers[5] = {10, 20, 30, 40, 50}; // Initializes an array with values
```

Programs:

Code 1:

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string grocery[5]={"Milk","Eggs","Cake","Biscuit","Candy"};
    for(int i=0;i<5;i++){
        cout<<grocery[i]<<endl;
    }
}
```

Output:

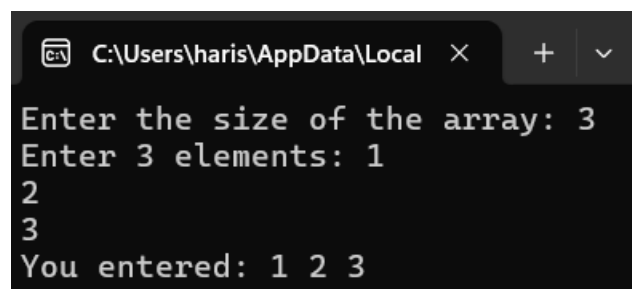


A screenshot of a C++ program's output. The window title is "C:\Users\haris\AppData\Local". The output displays a list of grocery items, each on a new line: Milk, Eggs, Cake, Biscuit, and Candy.

Code 2:

```
#include <iostream>
using namespace std;
int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int arr[size];
```

Output:



A screenshot of a C++ program's output. The window title is "C:\Users\haris\AppData\Local". The output shows the user being prompted to enter the size of the array, which is 3. Then, the user is prompted to enter 3 elements, and the numbers 1, 2, and 3 are entered. The final output is "You entered: 1 2 3".

```

cout << "Enter " << size << " elements: ";
for (int i = 0; i < size; i++) {
    cin >> arr[i];
}
cout << "You entered: ";
for (int i = 0; i < size; i++) {
    cout << arr[i] << " ";
}
}

```

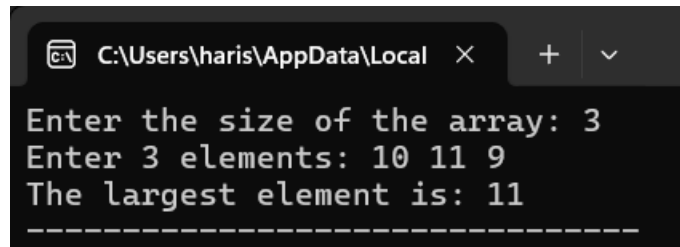
Code 3:

```

#include <iostream>
using namespace std;
int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int arr[size];
    cout << "Enter " << size << " elements: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    cout << "The largest element is: " << max;
    return 0;
}

```

Output:



```

C:\Users\haris\AppData\Local
Enter the size of the array: 3
Enter 3 elements: 10 11 9
The largest element is: 11
-----

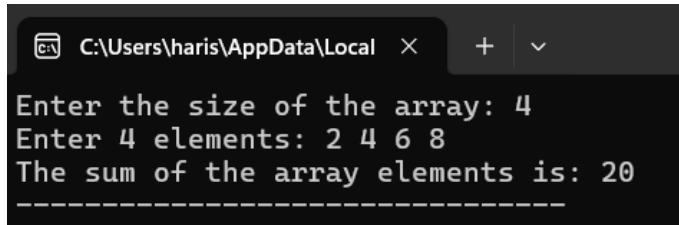
```

```
}
```

Code 4:

```
#include <iostream>
using namespace std;
int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int arr[size];
    cout << "Enter " << size << " elements: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    cout << "The sum of the array elements is: " << sum;
    return 0;
}
```

Output:

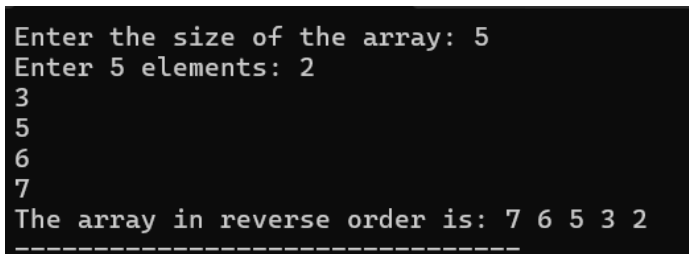


```
C:\Users\haris\AppData\Local
Enter the size of the array: 4
Enter 4 elements: 2 4 6 8
The sum of the array elements is: 20
-----
```

Code 5:

```
#include <iostream>
using namespace std;
int main() {
    int size;
    cout << "Enter the size of the array: ";
    cin >> size;
    int arr[size];
    cout << "Enter " << size << " elements: ";
```

Output:



```
C:\Users\haris\AppData\Local
Enter the size of the array: 5
Enter 5 elements: 2
3
5
6
7
The array in reverse order is: 7 6 5 3 2
-----
```

```
for (int i = 0; i < size; i++) {  
    cin >> arr[i];  
}  
cout << "The array in reverse order is: ";  
for (int i = size - 1; i >= 0; i--) {  
    cout << arr[i] << " ";  
}  
return 0;  
}
```


LAB 3 : 2-D ARRAY

Definition:

A **2D array** is a collection of data elements organized in rows and columns, forming a matrix-like structure. It is essentially an array of arrays, where each element is identified by two indices: the row index and the column index.

Syntax:

```
int matrix[3][4]; // 2D array with 3 rows and 4 columns
```

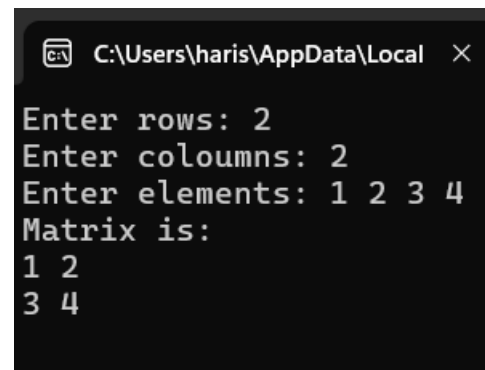
Programs:

Code 1:

```
#include<iostream>
using namespace std;
int main(){
    int n,m;
    cout<<"Enter rows: ";
    cin>>n;
    cout<<"Enter coloumns: ";
    cin>>m;
    int arr[n] [m];
    cout<<"Enter elements: ";
    for(int i=0;i<n;i++) {
        for (int j=0;j<m;j++) {
            cin>> arr[i][j];
        }
    }

    cout<<"Matrix is: "<<endl;
    for(int i=0;i<n;i++) {
```

Output:

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\haris\AppData\Local" and a close button. The output of the program is displayed in a monospaced font: "Enter rows: 2", "Enter coloumns: 2", "Enter elements: 1 2 3 4", "Matrix is:", followed by a 2x2 matrix of numbers: "1 2" on the first line and "3 4" on the second line.

```
C:\Users\haris\AppData\Local X
Enter rows: 2
Enter coloumns: 2
Enter elements: 1 2 3 4
Matrix is:
1 2
3 4
```

```

        for (int j=0; j<m;j++) {
            cout<<arr[i][j]<<" ";
        }

        cout<<"\n";
    }
}

```

Code 2:

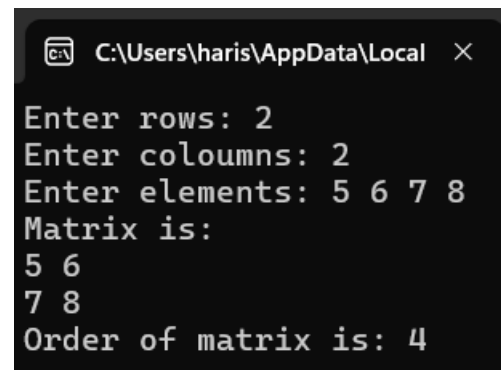
```

#include<iostream>
using namespace std;
int main(){
    int n,m,order;
    cout<<"Enter rows: ";
    cin>>n;
    cout<<"Enter coloumns: ";
    cin>>m;
    int arr[n] [m];
    cout<<"Enter elements: ";
    for(int i=0;i<n;i++) {
        for (int j=0;j<m;j++) {
            cin>> arr[i][j];
        }
    }

    cout<<"Matrix is: "<<endl;
    for(int i=0;i<n;i++) {
        for (int j=0; j<m;j++) {
            cout<<arr[i][j]<<" ";
        }
        cout<<"\n";
    }
}

```

Output:



```

Enter rows: 2
Enter coloumns: 2
Enter elements: 5 6 7 8
Matrix is:
5 6
7 8
Order of matrix is: 4

```

```

order= n*m;

cout<<"Order of matrix is: "<<order;    }

```

Code 3:

```

#include <iostream>

using namespace std;

int main(){
    int n,m;
    int mx=INT_MIN;
    cout<<"Enter rows: ";
    cin>>n;
    cout<<"Enter coloumns: ";
    cin>>m;
    int arr[n] [m];
    cout<<"Enter elements: ";
    for(int i=0;i<n;i++) {
        for (int j=0;j<m;j++) {
            cin>> arr[i][j];        }
    }

    cout<<"Matrix is: "<<endl;
    for(int i=0;i<n;i++) {
        for (int j=0; j<m;j++) {
            cout<<arr[i][j]<<" ";        }
        cout<<"\n";        }
    for(int i=0;i<n;i++) {
        for (int j=0; j<m;j++) {
            mx=max(mx,arr[i][j]);        }        }
    cout << "Maximum number is: " << mx << endl;    }

```

Output:

```

Enter rows: 2
Enter coloumns: 2
Enter elements: 3
8
9
2
Matrix is:
3 8
9 2
Maximum number is: 9

```

Code 4:

```
#include <iostream>

using namespace std;

int main(){

    int n,m;

    int mn=INT_MAX;

    cout<<"Enter rows: ";

    cin>>n;

    cout<<"Enter coloumns: ";

    cin>>m;

    int arr[n] [m];

    cout<<"Enter elements: ";

    for(int i=0;i<n;i++) {

        for (int j=0;j<m;j++) {

            cin>> arr[i][j];        }

    }

    cout<<"Matrix is: "<<endl;

    for(int i=0;i<n;i++) {

        for (int j=0; j<m;j++) {

            cout<<arr[i][j]<<" ";        }

        cout<<"\n";

    }

    for(int i=0;i<n;i++) {

        for (int j=0; j<m;j++) {

            mn=min(mn,arr[i][j]);        }

    }

    cout << "Minimum number is: " << mn << endl;    }
```

Output:

```
Enter rows: 2
Enter coloumns: 2
Enter elements: 9
7
6
1
Matrix is:
9 7
6 1
Minimum number is: 1
```

Code 5:

```
#include <iostream>

using namespace std;

int main()
{
    int rows, cols;

    cout << "Enter the number of rows and columns: ";

    cin >> rows >> cols;

    int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols];

    cout << "Enter elements of the first matrix:" << endl;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix1[i][j];
        }
    }

    cout << "Enter elements of the second matrix:" << endl;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> matrix2[i][j];
        }
    }

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }

    cout << "The sum of the matrices is:" << endl;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << sum[i][j] << " ";
        }
        cout << endl;
    }
}
```

Output:

```
Enter the number of rows and columns: 2 3
Enter elements of the first matrix:
1 3 5 7 9 8
Enter elements of the second matrix:
2 4 6 3 1 5
The sum of the matrices is:
3 7 11
10 10 13
```

LAB 4: VECTOR

Definition:

a **vector** is part of the Standard Template Library (STL) and is a **dynamic array** that can resize itself automatically when elements are added or removed. Unlike arrays, vectors can grow or shrink dynamically, and they provide additional functionality like bounds checking and various utility methods.

Syntax:

```
vector<int> vec(5); // Vector of size 5, all elements initialized to 0 (for int type)
```

Programs

Code 1:

```
#include<iostream>

#include<vector>

using namespace std;

int main(){

    vector <int>scores={68,98,75,53,62};

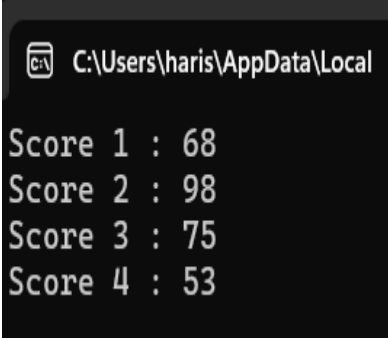
    for(int i=0;i<4;i++){

        cout<<"Score "<<i+1<<" : "<<scores[i]<<endl;

    }

}
```

Output:



```
C:\Users\haris\AppData\Local

Score 1 : 68
Score 2 : 98
Score 3 : 75
Score 4 : 53
```

Code 2:

```
#include<iostream>

#include<vector>

using namespace std;

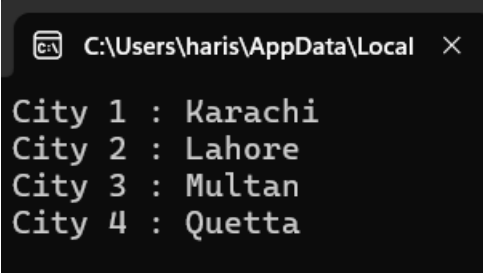
int main()

{

    vector <string>Cities={"Karachi","Lahore","Multan","Quetta"};

    for(int i=0;i<4;i++){
```

Output:



```
C:\Users\haris\AppData\Local

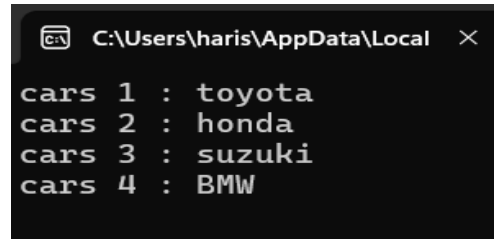
City 1 : Karachi
City 2 : Lahore
City 3 : Multan
City 4 : Quetta
```

```
cout<<"City "<<i+1<<" : "<<Cities[i]<<endl;    }    }
```

Code 3:

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
    vector <string>cars={"toyota","honda","suzuki","BMW"};
    for(int i=0;i<4;i++){
        cout<<"cars "<<i+1<<" : "<<cars[i]<<endl; }
}
```

Output:

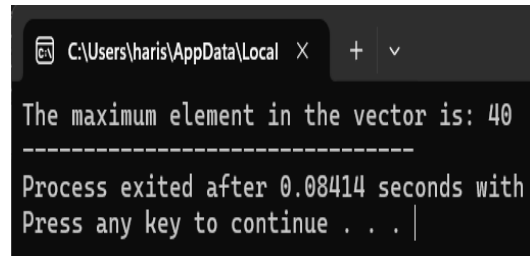


```
C:\Users\haris\AppData\Local
cars 1 : toyota
cars 2 : honda
cars 3 : suzuki
cars 4 : BMW
```

Code 4:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
    vector<int> vec = {10, 20, 5, 40, 15};
    int maxElement = *max_element(vec.begin(), vec.end());
    cout << "The maximum element in the vector is: " << maxElement;
    return 0;
}
```

Output:

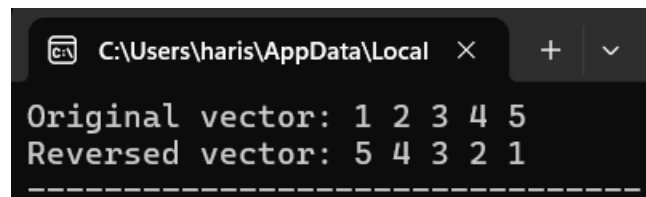


```
C:\Users\haris\AppData\Local
The maximum element in the vector is: 40
-----
Process exited after 0.08414 seconds with
Press any key to continue . . . |
```

Code 5:

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

Output:



```
C:\Users\haris\AppData\Local
Original vector: 1 2 3 4 5
Reversed vector: 5 4 3 2 1
-----
```

```
int main() {  
    vector<int> vec = {1, 2, 3, 4, 5};  
    cout << "Original vector: ";  
    for (int x : vec) {  
        cout << x << " ";  
    }  
    reverse(vec.begin(), vec.end());  
    cout << "\nReversed vector: ";  
    for (int x : vec) {  
        cout << x << " ";  
    }  
}
```


LAB 5: LIST

Definition:

A **list** is a **collection of ordered elements** of the same type, where each element is stored sequentially in memory or logically linked.

Syntax:

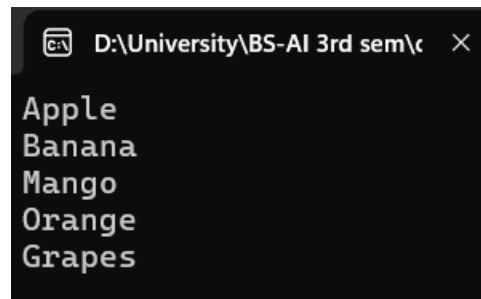
```
list<int> numbers; // Declares a list of integers
```

Programs

Code 1:

```
//fruit list
#include<iostream>
#include<string>
using namespace std;
int main(){
    string fruits[5]={"Apple","Banana","Mango","Orange","Grapes"};
    for(int i=0;i<5;i++){
        cout<<fruits[i]<<endl;    }
}
```

Output:

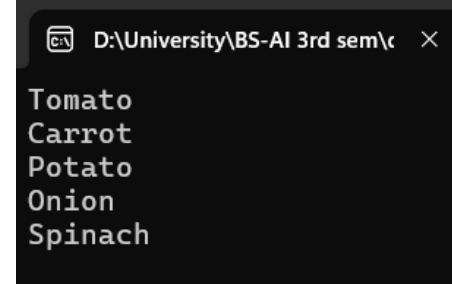


```
D:\University\BS-AI 3rd sem\c  X
Apple
Banana
Mango
Orange
Grapes
```

Code 2:

```
// vegetable list
#include<iostream>
#include<string>
using namespace std;
int main(){
    string vegetables[5]={"Tomato","Carrot","Potato","Onion","Spinach"};
    for(int i=0;i<5;i++){
        cout<<vegetables[i]<<endl;    }
}
```

Output:



```
D:\University\BS-AI 3rd sem\c  X
Tomato
Carrot
Potato
Onion
Spinach
```

Code 3:

```
// country list
#include<iostream>
#include<string>
using namespace std;
int main()    {
    string countries[5]={"USA","Canada","UK","Australia","Germany"};
    for(int i=0;i<5;i++)    {
        cout<<countries[i]<<endl;    }
}
```

Output:

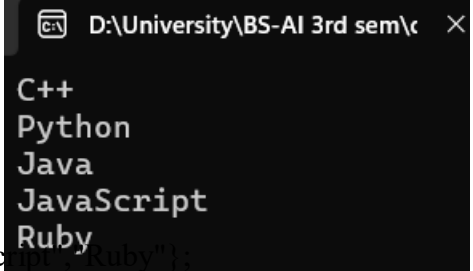


A screenshot of a C++ program's output. The window title is "D:\University\BS-AI 3rd sem\c". The output displays a list of five countries, each on a new line: USA, Canada, UK, Australia, and Germany.

Code 4:

```
#include<iostream>
#include<string>
using namespace std;
int main(){
    string languages[5]={"C++","Python","Java","JavaScript","Ruby"};
    for(int i=0;i<5;i++){
        cout<<languages[i]<<endl;    }
}
```

Output:

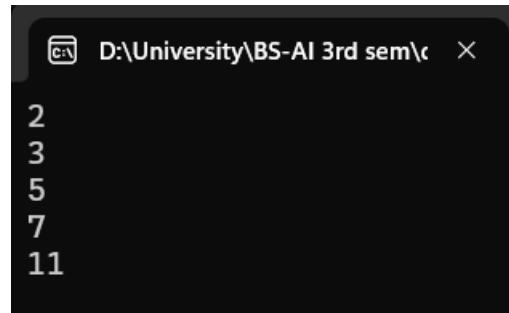


A screenshot of a C++ program's output. The window title is "D:\University\BS-AI 3rd sem\c". The output displays a list of five programming languages, each on a new line: C++, Python, Java, JavaScript, and Ruby.

Code 5:

```
#include<iostream>
using namespace std;
int main(){
    int primes[5] = {2, 3, 5, 7, 11};
    for(int i = 0; i < 5; i++){
        cout << primes[i]<<endl;    }    }
```

Output:



A screenshot of a C++ program's output. The window title is "D:\University\BS-AI 3rd sem\c". The output displays a list of five prime numbers, each on a new line: 2, 3, 5, 7, and 11.

Lab 6: Stack

- **Definition:**

A stack is a data structure that stores a collection of elements following the Last In, First Out (LIFO) principle. Elements can only be added or removed from the top of the stack. Stacks allow efficient insertion and deletion operations but restrict access to elements based on their position in the stack.

- **Syntax:**

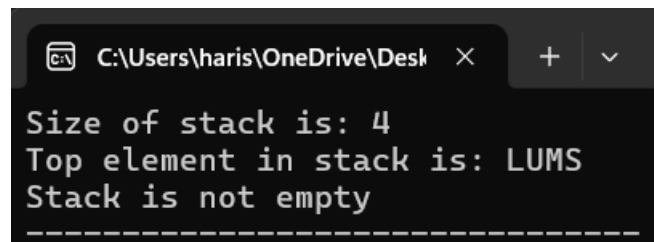
`stack<data-type> variable_name;`

PROGRAMS:

Code 1:

```
#include<iostream>
#include<string>
#include<stack>
using namespace std;
int main(){
    stack<string>universities;
    universities.push("TUF");
    universities.push("FAST");
    universities.push("NUST");
    universities.push("LUMS");
    universities.push("NTU");
    universities.push("Riphah");
    universities.push("BNU");
    universities.pop();
    universities.pop();
    universities.pop();
    cout<<"Size of stack is: "<<universities.size()<<"\n";
```

Output

A screenshot of a C++ program's output in a terminal window. The window title is "C:\Users\haris\OneDrive\Desktop". The output text is: "Size of stack is: 4", "Top element in stack is: LUMS", "Stack is not empty", followed by a dashed line.

```
C:\Users\haris\OneDrive\Desktop  X  +  v
Size of stack is: 4
Top element in stack is: LUMS
Stack is not empty
-----
```

```

cout<<"Top element in stack is: "<<universities.top();

if(universities.empty()==true){

    cout<<"\nStack is empty";

}

else

    cout<<"\nStack is not empty";}

```

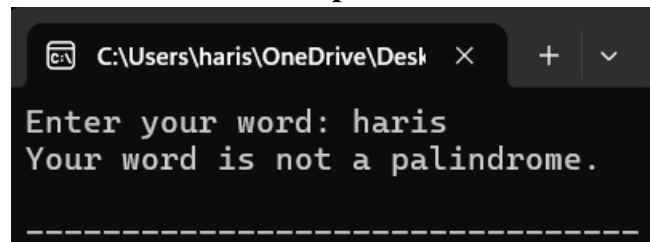
Code 2:

```

#include<iostream>
#include<stack>
using namespace std;
int main(){
    string word;
    cout<<"Enter your word: ";
    cin>>word;
    stack<char> x;
    for(char c : word) x.push(c);
    string rev;
    while(!x.empty()) {
        rev += x.top();
        x.pop();
    }
    if(word == rev)
        cout <<"Your word is a palindrome."<<endl;
    else
        cout<<"Your word is not a palindrome."<<endl;
}

```

Output:



A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\haris\OneDrive\Desktop'. The prompt displays the text 'Enter your word: haris' followed by 'Your word is not a palindrome.' and a line of dashes.

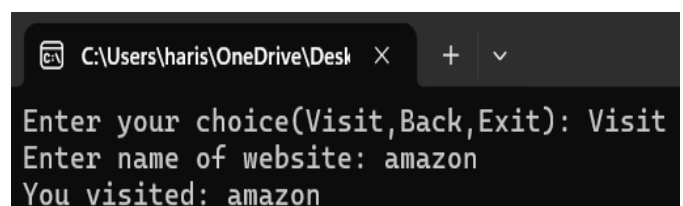
Code 3:

```

#include<iostream>
#include<stack>
using namespace std;
int main(){
    stack<string> browser;
    string choice;
    cout<<"Enter your choice(Visit,Back,Exit): ";
    cin>>choice;
    while (true) {
        if(choice=="Visit"){
            string name;

```

Output



A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\haris\OneDrive\Desktop'. The prompt displays the text 'Enter your choice(Visit,Back,Exit): Visit', 'Enter name of website: amazon', and 'You visited: amazon'.

```

        cout<<"Enter name of website: ";
        cin>>name;
        browser.push(name);
        cout << "You visited: " << name << endl;
    }
    else if (choice == "Back") {
        if (!browser.empty()) {
            cout << "Going back from: " << browser.top() << endl;
            browser.pop();
            if (!browser.empty()) {
                cout << "Current page: " << browser.top() << endl;
            } else {
                cout << "No more history. You're on a blank page.\n";
            }
        } else {
            cout << "No history to go back to.\n";
        }
    }

    } else if (choice == "Exit") {
        break;

    } else {
        cout << "Invalid command. Please enter 'visit', 'back', or 'exit'. \n";
    }
}

return 0;
}

```

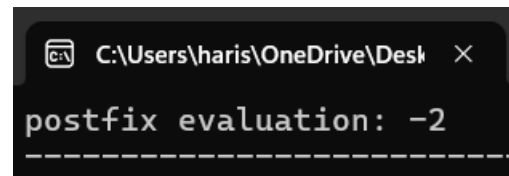
Code 4:

```

#include <iostream>
#include <stack>
using namespace std;
int evaluatePostfix(string exp)
{
    stack<int> st;
    for (int i = 0; i < exp.size(); ++i) {
        if (isdigit(exp[i]))
            st.push(exp[i] - '0');
        else {
            int val1 = st.top();
            st.pop();
            int val2 = st.top();
            st.pop();

```

Output



```

C:\Users\haris\OneDrive\Desktop ×
postfix evaluation: -2
-----

```

```

switch (exp[i]) {
case '+':
    st.push(val2 + val1);
    break;
case '-':
    st.push(val2 - val1);
    break;
case '*':
    st.push(val2 * val1);
    break;
case '/':
    st.push(val2 / val1);
    break;
}
}
}
return st.top();
}

int main()
{
    string exp = "231*+9-";
    cout << "postfix evaluation: " << evaluatePostfix(exp);
    return 0;
}

```

Code 5:

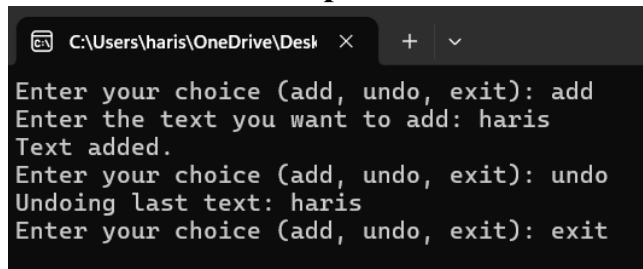
```

#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<string> commandStack;
    string choice;
    while (true) {
        cout << "Enter your choice (add, undo, exit): ";
        cin >> choice;

        if (choice == "add") {
            string addText;
            cout << "Enter the text you want to add: ";
            cin.ignore();
            cin >> addText;
            commandStack.push(addText);
            cout << "Text added.\n";

```

Output:



```

C:\Users\haris\OneDrive\Desktop
Enter your choice (add, undo, exit): add
Enter the text you want to add: haris
Text added.
Enter your choice (add, undo, exit): undo
Undoing last text: haris
Enter your choice (add, undo, exit): exit

```

```
} else if (choice == "undo") {  
    if (!commandStack.empty()) {  
        cout << "Undoing last text: " << commandStack.top() << endl;  
        commandStack.pop();  
    } else {  
        cout << "No text to undo.\n";  
    }  
  
} else if (choice == "exit") {  
    break;  
  
} else {  
    cout << "Invalid choice. Please enter 'add', 'undo', or 'exit'. \n";  
}  
}  
}
```

Lab 7: Queue

- **Definition:**

A queue is a data structure that stores a collection of elements following the **First In, First Out (FIFO)** principle. Elements are added at the back (enqueue) and removed from the front (dequeue). Queues allow efficient insertion and deletion operations while maintaining the order in which elements were added

- **Syntax:**

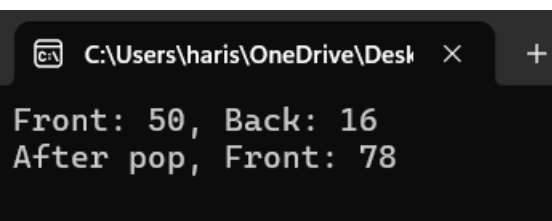
queue<data-type> variable_name;

Programs

Code 1:

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    queue<int> q;
    q.push(50);
    q.push(16);
    q.push(78);
    cout << "Front: " << q.front() << ", Back: " << q.back() << endl;
    q.pop();
    cout << "After pop, Front: " << q.front() << endl;
    return 0;
}
```

Output

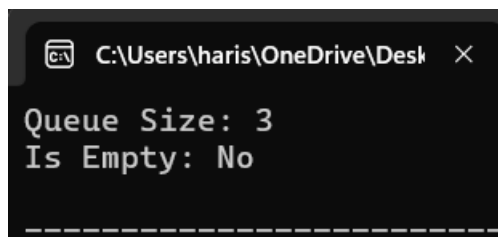


```
C:\Users\haris\OneDrive\Desktop >
Front: 50, Back: 16
After pop, Front: 78
```

Code 2:

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    queue<int> q;
    q.push(2); q.push(4); q.push(3);
    cout << "Queue Size: " << q.size() << endl;
    cout << "Is Empty: " << (q.empty() ? "Yes" : "No") << endl;
    return 0;
}
```

Output

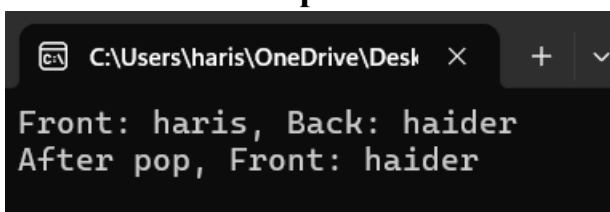


```
C:\Users\haris\OneDrive\Desktop >
Queue Size: 3
Is Empty: No
```

Code 3:

```
#include <iostream>
#include <queue>
```

Output:



```
C:\Users\haris\OneDrive\Desktop >
Front: haris, Back: haider
After pop, Front: haider
```



```

using namespace std;
int main()
{
    queue<string> q;
    q.push("haris");
    q.push("haider");
    cout << "Front: " << q.front() << ", Back: " << q.back() << endl;
    q.pop();
    cout << "After pop, Front: " << q.front() << endl;
    return 0;
}

```

Code 4:

```

#include <iostream>
#include <queue>
#include <stack>
using namespace std;
int main()
{

```

```

    queue<int> q; stack<int> s;
    for (int i = 1; i <= 5; ++i) q.push(i);
    while (!q.empty()) { s.push(q.front()); q.pop(); }
    while (!s.empty()) { cout << s.top() << " "; s.pop(); }
    return 0;
}

```

Code 5:

```

#include <iostream>
#include <queue>
using namespace std;
int main()
{

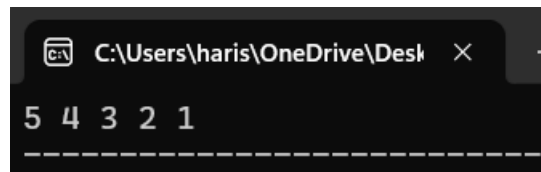
```

```

    queue<int> q1, q2;
    q1.push(1); q1.push(2);

```

Output:

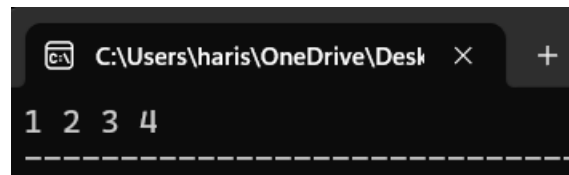


```

C:\Users\haris\OneDrive\Desktop >
5 4 3 2 1
-----

```

Output:



```

C:\Users\haris\OneDrive\Desktop >
1 2 3 4
-----

```

```
    q2.push(3); q2.push(4);  
    while (!q1.empty()) { cout << q1.front() << " "; q1.pop(); }  
    while (!q2.empty()) { cout << q2.front() << " "; q2.pop(); }  
    return 0;  
}
```

Lab 8: Quiz

1. Which of the following data structures is a collection of elements of the same type, stored at contiguous memory locations?

- A) Stack
- B) Queue
- C) Array
- D) List

2. In which data structure is the last element added the first to be removed?

- A) Queue
- B) Stack
- C) List
- D) Array

3. What is the term for accessing elements in an array by their index?

- A) Random Access
- B) Sequential Access
- C) Linear Access
- D) Binary Access

4. Which of the following allows dynamic resizing in C++?

- A) Array
- B) Vector
- C) Stack
- D) Queue

5. Which operation removes the first element in a queue?

- A) Push
- B) Pop

- C) Enqueue
- D) Dequeue

6. What is the default access order of elements in a stack?

- A) First-In, First-Out (FIFO)
- B) Last-In, First-Out (LIFO)
- C) Random
- D) Sequential

7. Which data structure allows both insertion and deletion at both ends?

- A) Array
- B) Stack
- C) List
- D) Deque

8. In an array of size n , what is the index of the last element?

- A) n
- B) $n-1$
- C) $n+1$
- D) 0

9. What is the primary difference between an array and a vector in C++?

- A) Arrays can grow dynamically
- B) Vectors can grow dynamically
- C) Vectors store elements in random order
- D) Arrays store elements in linked order

10. Which data structure would be most efficient for implementing an undo feature?

- A) Queue
- B) Stack
- C) Array
- D) List

11. What is the primary operation for adding an element to a stack?

- A) Insert
- B) Add
- C) Push
- D) Append

12. Which data structure follows the First-In, First-Out (FIFO) principle?

- A) Stack
- B) Queue
- C) Array
- D) Vector

13. In a vector, what function is used to add elements at the end?

- A) insert()
- B) append()
- C) push_back()
- D) add()

14. Which of the following is used to retrieve the number of elements in a vector?

- A) length()
- B) count()
- C) size()
- D) capacity()

15. What type of data structure is suitable for organizing tasks in the order they should be executed?

- A) Stack
- B) Queue
- C) Array
- D) List

16. Which data structure supports efficient access to any element by its index?

- A) Array
- B) Stack
- C) Queue
- D) Linked List

17. What operation is used to add elements to the front of a list?

- A) push_back()
- B) insert()
- C) push_front()
- D) append()

18. In a circular queue, if the front pointer is at the last position, where will it move after a dequeue operation?

- A) Remains at the same position
- B) Moves to the beginning
- C) Moves to the end
- D) Moves randomly

19. Which of the following data structures is a sequence container with dynamic size, allowing elements to be accessed by their position?

- A) Queue
- B) Stack
- C) Vector
- D) Linked List

20. What is the time complexity of accessing an element in an array by its index?

- A) $O(n)$
- B) $O(\log n)$
- C) $O(1)$ +
- D) $O(n^2)$

Lab 9: Deque

- **Definition:**

A deque is a data structure that stores a collection of elements and allows insertion and deletion from both ends (front and back). Deques provide flexibility for accessing elements while maintaining an ordered sequence. They are ideal for scenarios requiring dynamic resizing and dual-end operations.

- **Syntax:**

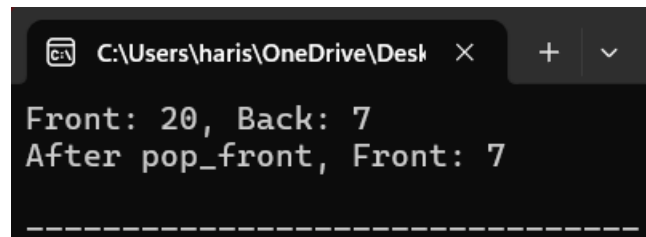
`deque<data-type> variable_name;`

Programs

Code 1:

```
#include <iostream>
#include <deque>
using namespace std;
int main()
{
    deque<int> dq;
    dq.push_back(7);
    dq.push_front(20);
    cout << "Front: " << dq.front() << ", Back: " << dq.back() << endl;
    dq.pop_front();
    cout << "After pop_front, Front: " << dq.front() << endl;
    return 0;
}
```

Output:

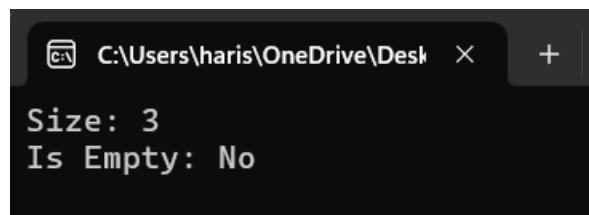


```
C:\Users\haris\OneDrive\Desk  X + v
Front: 20, Back: 7
After pop_front, Front: 7
-----
```

Code 2:

```
#include <iostream>
#include <deque>
using namespace std;
int main()
{
```

Output:



```
C:\Users\haris\OneDrive\Desk  X +
Size: 3
Is Empty: No
```

```

deque<int> dq = {1, 2, 3};

cout << "Size: " << dq.size() << endl;

cout << "Is Empty: " << (dq.empty() ? "Yes" : "No") << endl;

return 0;

}

```

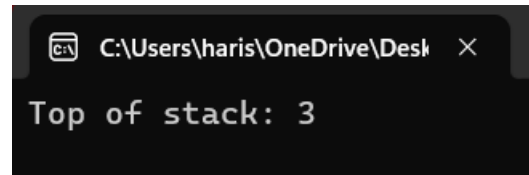
Code 3:

```

#include <iostream>
#include <deque>
using namespace std;
int main()
{
    deque<int> dq;
    dq.push_back(3);
    dq.push_back(1);
    dq.pop_back();
    cout << "Top of stack: " << dq.back() << endl;
    return 0;
}

```

Output



A screenshot of a terminal window with a dark background. The title bar shows the file path 'C:\Users\haris\OneDrive\Desktop' and a close button. The terminal output is 'Top of stack: 3'.

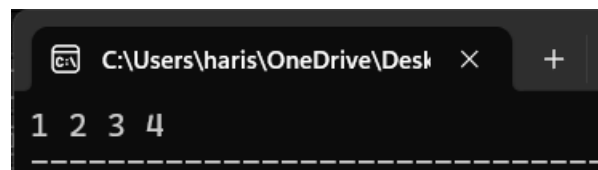
Code 4:

```

#include <iostream>
#include <deque>
using namespace std;
int main()
{
    deque<int> dq = {1, 2, 3, 4};
    for (int x : dq) cout << x << " ";
    return 0;
}

```

Output



A screenshot of a terminal window with a dark background. The title bar shows the file path 'C:\Users\haris\OneDrive\Desktop' and buttons for close and maximize. The terminal output is '1 2 3 4'.

Code 5:

```
#include <iostream>

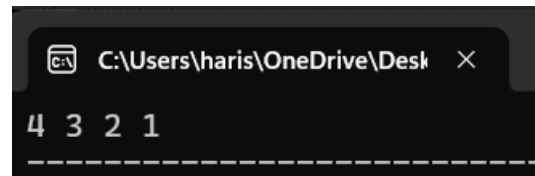
#include <deque>

#include <algorithm>

using namespace std;

int main() {
    deque<int> dq = {1, 2, 3, 4};
    reverse(dq.begin(), dq.end());
    for (int x : dq) cout << x << " ";
    return 0;
}
```

Output

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\haris\OneDrive\Desktop" and a close button. The command prompt displays the output of the program: "4 3 2 1".

```
C:\Users\haris\OneDrive\Desktop >
4 3 2 1
```

LAB 10: Single Link list

Definition:

A Linked List is a linear data structure where elements, called nodes, are connected using pointers. Each node contains two parts.

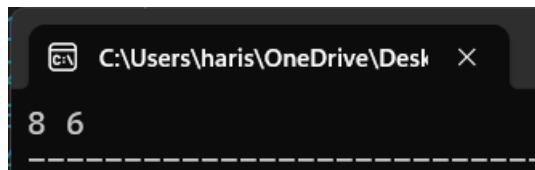
- Singly link list: A singly link list is the link where each node has a single pointer to the next node. It Traverses is one-directional and the last node's pointer is nullptr.

Programs

Code 1:

```
#include<iostream>
using namespace std;
class node{
public:
    int data;
    node *link;
    node *head;
    node *tail;
    node *current;
    node (){
        head = nullptr;
        tail = nullptr;
    }
    void create(int data){
        if(head == nullptr){
            node *n = new node();
            n->data = data;
            n->link = nullptr;
            head = tail = n;
```

Output:



```

    }
    else{
        node *n = new node();
        n->data = data;
        n->link = nullptr;
        tail->link=n;
        tail = n;
    }
}

void insertAtBegin(int data) {
    node *n = new node();
    n->data = data;
    n->link = head;
    head = n;
}

void display(){
    current = head;
    while(current!=nullptr){
        cout<<current->data<<" ";
        current = current->link;
    }
}

};

int main(){
    node obj;
    obj.create(6);
    obj.insertAtBegin(8);
    obj.display();
}

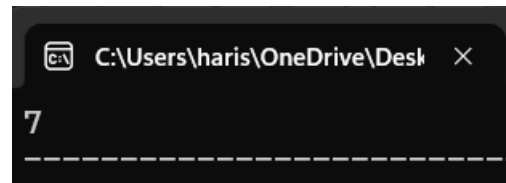
```

```
}
```

Code 2:

```
#include<iostream>
using namespace std;
class node{
public:
    int data;
    node *link;
    node *head;
    node *tail;
    node *current;
    node(){
        head = nullptr;
        tail = nullptr;
    }
    void create(int data){
        if(head == nullptr){
            node *n = new node();
            n->data = data;
            n->link = nullptr;
            head = tail = n;
        }
        else{
            node *n = new node();
            n->data = data;
            n->link = nullptr;
            tail->link=n;
            tail = n;
        }
    }
};
```

Output:



```

    }
}

void deleteAtBegin() {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
    node *temp = head;
    head = head->link;
    delete temp;
    if (head == nullptr) { // If the list is now empty, set tail to nullptr
        tail = nullptr;
    }
}

void display()
{
    current = head;
    while(current!=nullptr){
        cout<<current->data<<" ";
        current = current->link;
    }
}

};

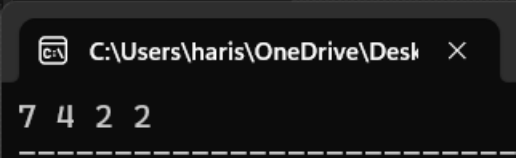
int main(){
    node haris;
    haris.create(7);
    haris.display();
}

```

Code 3:

```
#include<iostream>
using namespace std;
class node{
    public:
    int data;
    node *link;
    node *head;
    node *tail;
    node *current;
    node (){
        head = nullptr;
        tail = nullptr;
    }
    void create(int data){
        if(head == nullptr){
            node *n = new node();
            n->data = data;
            n->link = nullptr;
            head = tail = n;
        }
        else{
            node *n = new node();
            n->data = data;
            n->link = nullptr;
            tail->link=n;
            tail = n;
        }
    }
};
```

Output:



A screenshot of a C++ program's output. The window title is "C:\Users\haris\OneDrive\Desktop". The output shows the numbers "7 4 2 2" on a single line. The window has a standard Windows interface with a close button (X) in the top right corner.

```

    }
}

void insertAtPosition(int data, int position) {
    node *n = new node();
    n->data = data;
    if (position == 0) { // Insert at the beginning
        n->link = head;
        head = n;
        if (tail == nullptr) {
            tail = n;
        }
    } else {
        current = head;
        for (int i = 0; i < position - 1 && current != nullptr; ++i) {
            current = current->link;
        }
        if (current != nullptr) {
            n->link = current->link;
            current->link = n;
            if (n->link == nullptr) {
                tail = n;
            }
        } else {
            cout << "Position out of bounds" << endl;
        }
    }
}

void display(){

```

```

        current = head;
        while(current!=nullptr){
            cout<<current->data<<" ";
            current = current->link;
        }
    }
};

int main(){
    node haris;
    haris.create(7);
    haris.create(4);
    haris.create(2);
    haris.insertAtPosition(2,2);
    haris.display();
}

```

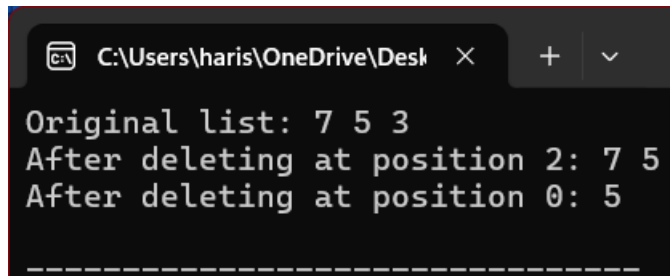
Code 4:

```

#include<iostream>
using namespace std;
class node {
public:
    int data;
    node *link;
    node *head;
    node *tail;
    node *current;
    node() {
        head = nullptr;
        tail = nullptr;
    }
};

```

Output:



```

C:\Users\haris\OneDrive\Desktop
Original list: 7 5 3
After deleting at position 2: 7 5
After deleting at position 0: 5
-----

```



```

}

void create(int data) {
    node *n = new node();
    n->data = data;
    n->link = nullptr;
    if (head == nullptr) {
        head = tail = n;
    } else {
        tail->link = n;
        tail = n;
    }
}

void deleteAtBegin() {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
    node *temp = head;
    head = head->link;
    if (head == nullptr) { // List had only one node, so update tail
        tail = nullptr;
    }
    delete temp;
}

void deleteAtPosition(int position) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }

```

```

    }

    if (position == 0) { // Special case for deleting the head
        deleteAtBegin();

        return;
    }

    current = head;

    for (int i = 0; i < position - 1 && current->link != nullptr; ++i) {
        current = current->link;
    }

    if (current->link == nullptr) {
        cout << "Position out of bounds." << endl;
    } else {
        node *temp = current->link;
        current->link = temp->link;

        if (temp->link == nullptr) { // Update tail if we're deleting the last node
            tail = current;
        }

        delete temp;
    }
}

void display() {
    current = head;

    while (current != nullptr) {
        cout << current->data << " ";

        current = current->link;
    }

    cout << endl;
}

```

```

};

int main() {
    node haris;
    haris.create(7);
    haris.create(5);
    haris.create(3);
    cout << "Original list: ";
    haris.display();
    haris.deleteAtPosition(2);
    cout << "After deleting at position 2: ";
    haris.display();
    haris.deleteAtPosition(0);
    cout << "After deleting at position 0: ";
    haris.display();
    return 0;
}

```

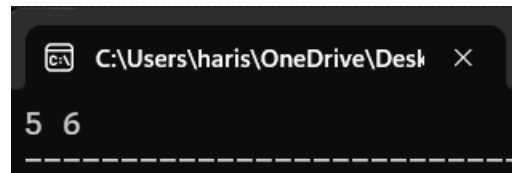
Code 5:

```

#include<iostream>
using namespace std;
class node{
public:
    int data;
    node *link;
    node *head;
    node *tail;
    node *current;
    node (){
        head = nullptr;

```

Output:



```

    tail = nullptr;
}

void create(int data){
    if(head == nullptr){
        node *n = new node();
        n->data = data;
        n->link = nullptr;
        head = tail = n;
    }
    else{
        node *n = new node();
        n->data = data;
        n->link = nullptr;
        tail->link=n;
        tail = n;
    }
}

void insertAtEnd(int data) {
    node *n = new node();
    n->data = data;
    n->link = nullptr;
    if (head == nullptr) {
        head = tail = n;
    } else {
        tail->link = n;
        tail = n;
    }
}

```

```
void display(){
    current = head;
    while(current!=nullptr){
        cout<<current->data<<" ";
        current = current->link;
    }
}

};

int main(){
    node haris;
    haris.create(5);
    haris.insertAtEnd(6);
    haris.display();
}
```

LAB 11: Double Link list

Definition:

A doubly linked list is a data structure where each node contains data, a pointer to the next node, and a pointer to the previous node. This allows traversal of the list in both forward and backward directions.

• Characteristics

1. Each node has pointers to both the previous and next nodes.
2. Traversal is bidirectional.
3. The head's prev pointer and the last node's next pointer are nullptr.

Programs:

Code 1:

```
struct DoublyNode {  
    int data;  
  
    DoublyNode* prev;  
    DoublyNode* next;  
  
    DoublyNode(int val) : data(val), prev(nullptr), next(nullptr) {}  
};
```

Output:

```
10 <--> 20 <--> 30 <--> nullptr  
-----
```

Code 2:

```
void insertEnd(DoublyNode*& head, int val) {  
    DoublyNode* newNode = new DoublyNode(val);  
  
    if (head == nullptr) {  
        head = newNode;  
  
        return;  
    }  
  
    DoublyNode* temp = head;  
  
    while (temp->next != nullptr) {  
        temp = temp->next;  
    }  
}
```

Output:

```
10 <--> 20 <--> 30 <--> nullptr  
-----
```

```
temp->next = newNode;
newNode->prev = temp;}
```

Code 3:

```
void deleteNode(DoublyNode*& head, int val) {
    if (head == nullptr) return;
    if (head->data == val) {
        DoublyNode* temp = head;
        head = head->next;
        if (head != nullptr) head->prev = nullptr;
        delete temp;
        return;
    }
    DoublyNode* temp = head;
    while (temp != nullptr && temp->data != val) {
        temp = temp->next;
    }
    if (temp == nullptr) return;
    if (temp->next != nullptr) temp->next->prev = temp->prev;
    if (temp->prev != nullptr) temp->prev->next = temp->next;
    delete temp;
}
```

Output:

```
10 <--> 20 <--> nullptr
20 <--> nullptr
nullptr
```

Code 4:

```
void display(DoublyNode* head) {
    DoublyNode* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " <--> ";
        temp = temp->next;
    }
}
```

Output:


```
10 <--> 20 <--> 30 <--> nullptr
```

```
cout << "nullptr" << endl;}
```

Code 5:

```
void insertatposition(int position,int data){
    node *n = new node();
    n->data=data;
    if(position==1){
        n->next=head;
        if(head!=NULL){
            head->prev=n;}
        head = n;
        if(tail==NULL){
            tail=n;}}
    else{
        current = head;
        int count = 1;
        while(current!=nullptr && count<position-1){
            current = current->next;
            count++;}
        if(current==NULL){
            cout<<"position out of bounds";
            delete n;
            return;}
        n->next=current->next;
        n->prev=current;
        if (current->next != nullptr) {
            current->next->prev = n;} else { // If inserting at the tail
            tail = n;}
        current->next = n;}}
```

Output:



```
10 <-> 15 <-> 20 <-> nullptr
-----
```


LAB 12: Circular Link List

Definition:

A circular linked list is a type of linked list in which the last node points back to the first node, forming a loop. This structure allows traversal to continue indefinitely from any point in the list.

Characteristics

- The last node points to the first node.
- Can be singly or doubly linked.
- Enables circular traversal.

Programs

Code 1:

```
struct CNode {  
    int data;  
    CNode* next;  
    CNode(int val) : data(val), next(nullptr) {}  
};
```

Output:

```
10 20 30  
-----
```

Code 2:

```
void insertEnd(CNode*& head, int val) {  
    CNode* newNode = new CNode(val);  
    if (head == nullptr) {  
        head = newNode;  
        newNode->next = head;  
        return;  
    }  
    CNode* temp = head;  
    while (temp->next != head) {  
        temp = temp->next;  
    }
```

Output:

```
10 20 30  
-----
```

```
Process exited after 0.006958 seconds with return value 0
```

```

temp->next = newNode;
newNode->next = head;
}

```

Code 3:

```

void deleteNode(CNode*& head, int val) {
    if (head == nullptr) return;
    if (head->data == val && head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    CNode* temp = head;
    CNode* prev = nullptr;
    do {
        if (temp->data == val) break;
        prev = temp;
        temp = temp->next;
    } while (temp != head);
    if (temp == head && temp->data != val) return;
    if (temp == head) {
        prev = head;
        while (prev->next != head) prev = prev->next;
        head = head->next;
        prev->next = head;
    } else {
        prev->next = temp->next;
    }
    delete temp;
}

```

Output:

```

10 20 30
10 30
-----
Process exited after 0.062 seconds with return value 0

```

```

} DoublyNode* temp = head;

while (temp != nullptr && temp->data != val) {
    temp = temp->next;
}

if (temp == nullptr) return;

if (temp->next != nullptr) temp->next->prev = temp->prev;
if (temp->prev != nullptr) temp->prev->next = temp->next;

delete temp;
}

```

Code 4:

```

void display(CNode* head) {
    if (head == nullptr) return;
    CNode* temp = head;
    do {
        cout << temp->data << " -> ";
        temp = temp->next;
    } while (temp != head);
    cout << "(head)" << endl;
}

```

Output:

```

10 20 30
-----
Process exited after 0.06958 seconds with return value 0

```

Code 5:

```

void insertAtPosition(int data, int position) {
    node *n = new node();
    n->data = data;
    if (position == 0) { // Insert at the beginning
        n->link = head;
        head = n;
        if (tail == nullptr) {
            tail = n;

```

Output:

```

10 20 30
-----
Process exited after 0.06958 seconds with return value 0
Press any key to continue . . .

```

```
    }  
} else {  
    current = head;  
    for (int i = 0; i < position - 1 && current != nullptr; ++i) {  
        current = current->link;  
    }  
    if (current != nullptr) {  
        n->link = current->link;  
        current->link = n;  
        if (n->link == nullptr) {  
            tail = n;  
        }  
    }  
} else {  
    cout << "Position out of bounds" << endl;
```

Lab 13: Trees

- **Definition:**

A tree is a hierarchical data structure that organizes elements in a parent-child relationship. It starts with a single root node and branches into sub-nodes (children), forming levels. Each node can have zero or more children, and there is no limit to the depth of the hierarchy. Trees are widely used for efficient searching, sorting, and hierarchical data representation

- **Key Characteristics of a Tree:**

1. **Hierarchical Structure:** Organized in levels, with a root node and child nodes forming a hierarchy.
2. **Parent-Child Relationship:** Each node (except the root) has one parent and may have multiple children.
3. **Recursive Representation:** Trees are naturally represented and traversed using recursion (e.g., in-order, pre-order, post-order).

Programs

Code 1:

```
#include <iostream>

using namespace std;

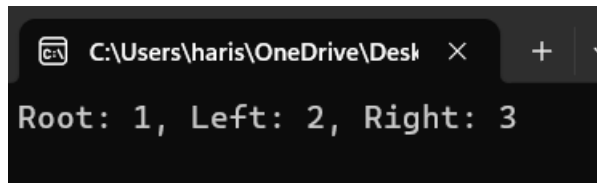
struct Node {
    int data;
    Node *left, *right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

int main()
{
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);

    cout << "Root: " << root->data << ", Left: " << root->left->data << ", Right: " << root->right->data << endl;

    return 0;
}
```

Output



The screenshot shows a terminal window with the output of the program. The text displayed is "Root: 1, Left: 2, Right: 3". The terminal window has a title bar that reads "C:\Users\haris\OneDrive\Desktop" and a close button.

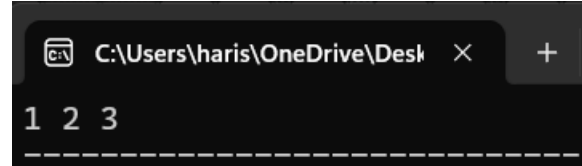
Code 2:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node *left, *right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

void preorder(Node* root) {
    if (!root) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    preorder(root);
    return 0;
}
```

Output

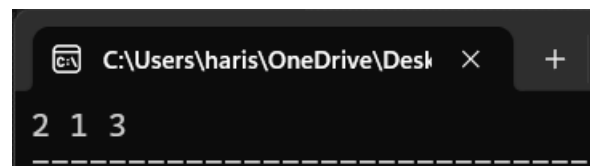


```
C:\Users\haris\OneDrive\Desktop  X  +
1 2 3
-----
```

Code 3:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node *left, *right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
}
```

Output



```
C:\Users\haris\OneDrive\Desktop  X  +
2 1 3
-----
```

```

};

void inorder(Node* root) {
    if (!root) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    inorder(root);
    return 0;
}

```

Code 4:

```

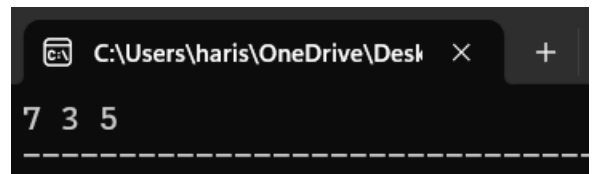
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node *left, *right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

void postorder(Node* root) {
    if (!root) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

int main() {
    Node* root = new Node(5);

```

Output



```

C:\Users\haris\OneDrive\Desktop  ×  +
7 3 5

```

```

    root->left = new Node(7);

    root->right = new Node(3);

    postorder(root);

    return 0;
}

```

Code 5:

```

#include <iostream>
#include <queue>

using namespace std;

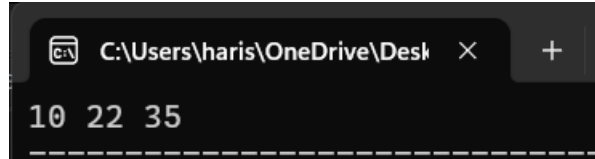
struct Node {
    int data;
    Node *left, *right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

void levelOrder(Node* root) {
    if (!root) return;
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        Node* curr = q.front();
        q.pop();
        cout << curr->data << " ";
        if (curr->left) q.push(curr->left);
        if (curr->right) q.push(curr->right);
    }
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(22);
    root->right = new Node(35);
    levelOrder(root);
}

```

Output



A screenshot of a terminal window with a dark background. The title bar shows the file path "C:\Users\haris\OneDrive\Desk" and window control buttons. The terminal displays the output "10 22 35" on a single line, followed by a dashed line at the bottom.

Lab 14: Binary Search Trees

- **Definition:**

A Binary Search Tree (BST) is a type of binary tree in which each node follows a specific ordering property:

1. **Left Subtree:** Contains nodes with keys less than the parent node's key.
2. **Right Subtree:** Contains nodes with keys greater than the parent node's key. This property makes BSTs highly efficient for searching, insertion, and deletion operations.

- **Key Characteristics of a Tree:**

1. **Ordered Structure:** Nodes are arranged in a way that enables efficient operations.
2. **Recursive Representation:** BST operations like traversal and manipulation are naturally implemented using recursion.
3. **No Duplicates:** In a standard BST, duplicate elements are not allowed.
4. **Traversals:** Common traversal methods include in-order, pre-order, and post-order, where in-order traversal yields a sorted sequence of elements.

Programs

Code 1:

```
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int value) :
data(value), left(nullptr), right(nullptr) {}
};

Node* insert(Node* root, int value) {
    if (root == nullptr) return new Node(value);
    if (value < root->data)
        root->left = insert(root->left, value);
    else
        root->right = insert(root->right, value);
    return root;
}
```

Output

```
In-order Traversal: 20 30 40 50 60 70 80
Pre-order Traversal: 50 30 20 40 70 60 80
Post-order Traversal: 20 40 30 60 80 70 50
Search 40: Found
Minimum value: 20
Maximum value: 80
Height of the tree: 2
In-order Traversal after deleting 30: 20 40 50 60 70 80
Is valid BST: Yes
Successor of 50: 60
Predecessor of 50: 40
```

Code 2:

```
#include <iostream>

using namespace std;

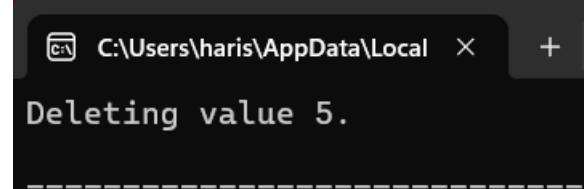
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};

Node* findMin(Node* root) {
    while (root && root->left != NULL) {
        root = root->left;
    }
    return root;
}

Node* deleteNode(Node* root, int key) {
    if (root == NULL) return root;

    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
```

Output

A screenshot of a terminal window with a dark background. The title bar shows the file path 'C:\Users\haris\AppData\Local' and standard window controls. The terminal displays the text 'Deleting value 5.' in a light-colored monospace font.

```

        Node* temp = root->right;
        delete root;
        return temp;
    } else if (root->right == NULL) {
        Node* temp = root->left;
        delete root;
        return temp;
    }
    Node* temp = findMin(root->right);
    root->data = temp->data;
    root->right = deleteNode(root->right, temp->data);
}
return root;
}

int main()
{
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    root->left->left = new Node(3);
    root->left->right = new Node(7);
    root->right->left = new Node(12);
    root->right->right = new Node(18);
    cout << "Deleting value 5." << endl;
    root = deleteNode(root, 5);
    return 0;
}

```

Code 3:

```
#include <iostream>

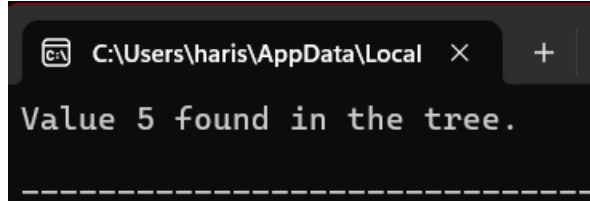
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};

bool search(Node* root, int key) {
    if (root == NULL) return false;
    if (root->data == key) return true;
    if (key < root->data) return search(root->left, key);
    return search(root->right, key);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    int searchKey = 5;
    if (search(root, searchKey)) {
        cout << "Value " << searchKey << " found in the tree." << endl;
    } else {
        cout << "Value " << searchKey << " not found in the tree." << endl;
    }
}
```

Output



A screenshot of a terminal window with a dark background. The title bar shows the file path "C:\Users\haris\AppData\Local" and standard window controls. The terminal displays the output "Value 5 found in the tree." in a light-colored monospace font.

```

    }
    return 0;
}

```

Code 4:

```

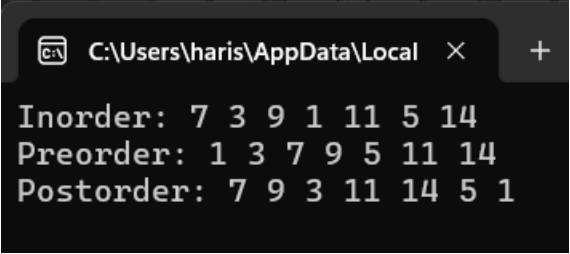
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};

void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

void preorder(Node* root) {
    if (root == NULL) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

```

Output



```

Inorder: 7 3 9 1 11 5 14
Preorder: 1 3 7 9 5 11 14
Postorder: 7 9 3 11 14 5 1

```

```


void postorder(Node* root)
{
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

int main()
{
    Node* root = new Node(1);
    root->left = new Node(3);
    root->right = new Node(5);
    root->left->left = new Node(7);
    root->left->right = new Node(9);
    root->right->left = new Node(11);
    root->right->right = new Node(14);
    cout << "Inorder: ";
    inorder(root);
    cout << endl;
    cout << "Preorder: ";
    preorder(root);
    cout << endl;
    cout << "Postorder: ";
    postorder(root);
    cout << endl;
    return 0;
}

```

Code 5:

```
int findMinValue(Node* root) {  
    while (root->left != nullptr) root = root->left;  
    return root->data;  
}  
  
int findMaxValue(Node* root) {  
    while (root->right != nullptr) root = root->right;  
    return root->data;  
}
```

Output

```
Minimum value: 20  
Maximum value: 80
```