



# LAB MANUALS

**Submitted by:** Yumna Irfan

**Registration no:** 2023-bs-ai-021

**Submitted to:** Miss Irsha Qureshi

**Department:** BS Artificial Intelligence

## Table of Contents

<b>Introduction to DSA.....</b>	<b>03</b>
<b>Lab 1: Array.....</b>	<b>04</b>
<b>Lab 2: 2DArray.....</b>	<b>08</b>
<b>Lab 3: Vectors.....</b>	<b>12</b>
<b>Lab 4: List.....</b>	<b>21</b>
<b>Lab 5: Stack.....</b>	<b>29</b>
<b>Lab 6: Queue.....</b>	<b>37</b>
<b>Lab 7: Single Link list.....</b>	<b>42</b>
<b>Lab 8: Circular Link list.....</b>	<b>61</b>
<b>Lab 9: Doubly Link list.....</b>	<b>78</b>
<b>Lab 10: Binay Search Tree.....</b>	<b>94</b>

# **Introductory Lab**

## **Data Structures:**

Data structures are ways of organizing and storing data to efficiently perform operations on it. The choice of data structure affects the performance of algorithms and the efficiency of the program as a whole.

### **Types:**

1. Linear data structures
2. Non-linear data structures (Trees, Graphs)

## **Linear data structures:**

Linear data structures are those in which the elements are stored sequentially, meaning that each element is connected to its previous and next element in a linear order. The primary characteristic of linear data structures is that they allow traversal in a single direction from one element to the next.

### **Types:**

1. Array
2. Link list
3. Stack
4. Queue

## **Non-linear data structures:**

Non-linear data structures are those in which the elements are not arranged in a sequential or linear order. In these structures, each element can have multiple relationships with other elements. Non-linear data structures are often used when the data requires more complex relationships between elements. These data structures allow for more efficient algorithms for certain types of problems, such as hierarchical or interconnected data.

### **Types:**

1. Trees
2. Graphs

# LAB 01

## Array

Arrays are a collection of elements, all of the same type, stored in contiguous memory locations.

### **Syntax:**

```
type arrayName[size];
```

---

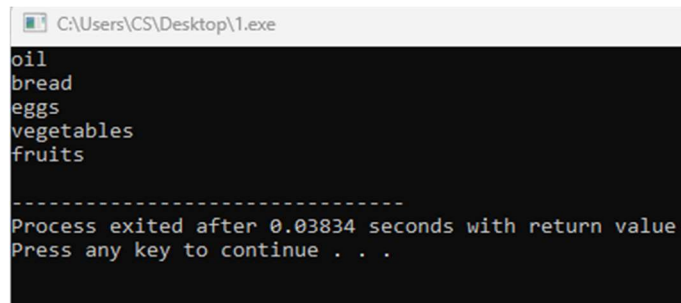
### **Code 1:**

```
#include<iostream>

using namespace std;

int main()
{
    string arr[5]={"oil" , "bread" , "eggs" , "vegetables" , "fruits"};
    for(int i=0 ; i<5 ; i++)
    {
        cout<<arr[i]<<endl;
    }
    return 0;
}
```

### **Output:**



```
C:\Users\CS\Desktop\1.exe
oil
bread
eggs
vegetables
fruits
-----
Process exited after 0.03834 seconds with return value
Press any key to continue . . .
```

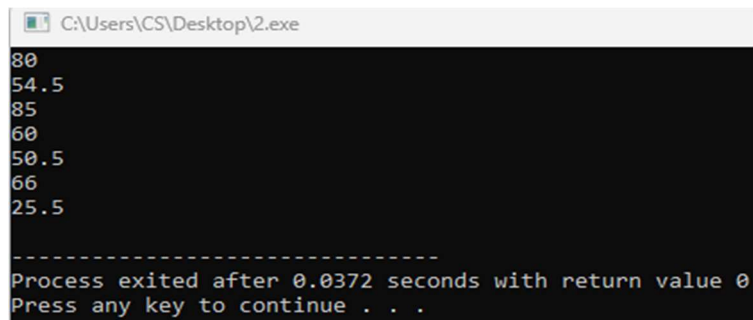
## Code 2:

```
#include<iostream>

using namespace std;

int main()
{
    float arr[7]={80 , 54.5 , 85 , 60 , 50.5 , 66 , 25.5 };
    for(int i=0 ; i<7 ; i++)
    {
        cout<<arr[i]<<endl;
    }
    return 0;
}
```

## Output:



```
C:\Users\CS\Desktop\2.exe
80
54.5
85
60
50.5
66
25.5
-----
Process exited after 0.0372 seconds with return value 0
Press any key to continue . . .
```

## Code 3:

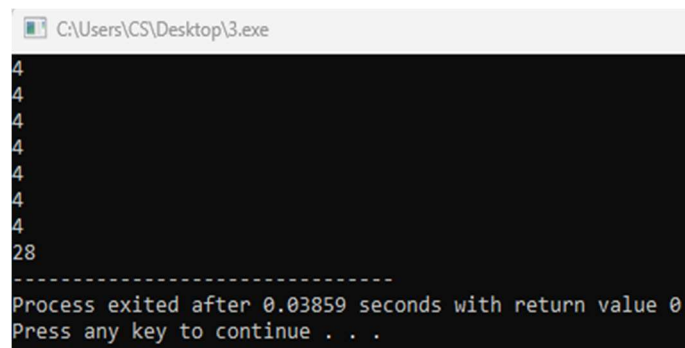
```
#include<iostream>

using namespace std;

int main()
{
    float arr[7]={80 , 54.5 , 85 , 60 , 50.5 , 66 , 25.5 };
    for(int i=0 ; i<7 ; i++)
    {
```

```
        cout<<sizeof(arr[i])<<endl;
    }
    cout<<sizeof(arr);
    return 0;
}
```

## Output:



```
C:\Users\CS\Desktop\3.exe
4
4
4
4
4
4
4
28
-----
Process exited after 0.03859 seconds with return value 0
Press any key to continue . . .
```

## Code 4:

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

int main() {
    string fruits[5];
    fruits[0] = "Apple";
    fruits[1] = "Banana";
    fruits[2] = "Cherry";
    fruits[3] = "Date";
    fruits[4] = "berry";

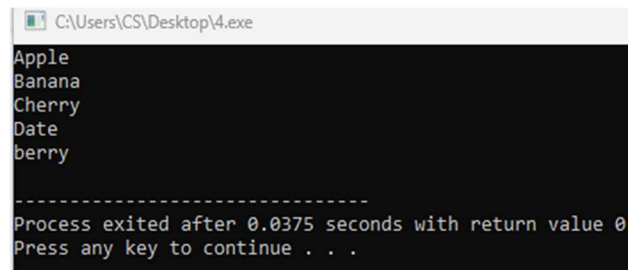
    for(int i = 0; i < 5; i++) {
```

```

        cout << fruits[i] << "\n";
    }
    return 0;
}

```

## Output:



```

C:\Users\CS\Desktop\4.exe
Apple
Banana
Cherry
Date
berry
-----
Process exited after 0.0375 seconds with return value 0
Press any key to continue . . .

```

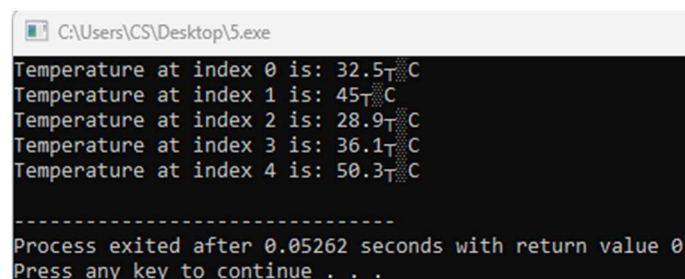
## Code 5:

```

#include <iostream>
using namespace std;
int main()
{
    double temperatures[5] = {32.5, 45.0, 28.9, 36.1, 50.3};
    for (int index = 0; index < 5; index++) {
        cout << "Temperature at index " << index << " is: " << temperatures[index] << "°C" << endl;
    }
    return 0;}

```

## Output:



```

C:\Users\CS\Desktop\5.exe
Temperature at index 0 is: 32.5°C
Temperature at index 1 is: 45°C
Temperature at index 2 is: 28.9°C
Temperature at index 3 is: 36.1°C
Temperature at index 4 is: 50.3°C
-----
Process exited after 0.05262 seconds with return value 0
Press any key to continue . . .

```

## **LAB 02**

### **Multi-Dimensional array**

A multi-dimensional array is an array that contains one or more arrays. It can be considered as an array of arrays, where each element itself is an array. The most common multi-dimensional arrays are 2D arrays (two-dimensional arrays), but they can have more than two dimensions (3D, 4D, etc.).

#### **Syntax:**

```
type array_name[row_size][column_size];
```

---

#### **Code 1:**

```
#include <iostream>

using namespace std;

int main()
{
    float matrix[2][3] = {{1.1, 2.2, 3.3}, {4.4, 5.5, 6.6}};

    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```



## Output:

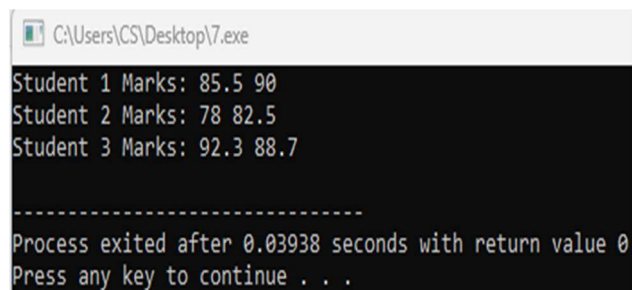


```
C:\Users\CS\Desktop\6.exe
1.1 2.2 3.3
4.4 5.5 6.6
-----
Process exited after 0.03842 seconds with return value 0
Press any key to continue . . .
```

## Code 2:

```
#include <iostream>
using namespace std;
int main() {
    float marks[3][2] = {{85.5, 90.0}, {78.0, 82.5}, {92.3, 88.7}};
    for (int i = 0; i < 3; i++) {
        cout << "Student " << i + 1 << " Marks: ";
        for (int j = 0; j < 2; j++) {
            cout << marks[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

## Output:



```
C:\Users\CS\Desktop\7.exe
Student 1 Marks: 85.5 90
Student 2 Marks: 78 82.5
Student 3 Marks: 92.3 88.7
-----
Process exited after 0.03938 seconds with return value 0
Press any key to continue . . .
```

### Code 3:

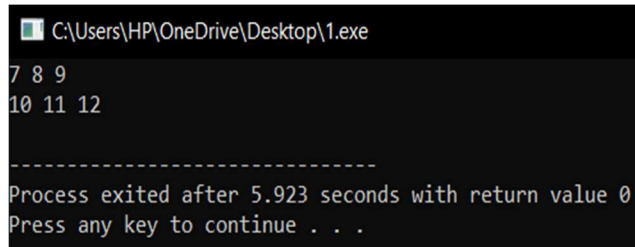
```
#include <iostream>

using namespace std;

int main()
{
    int myArray[2][3] = {{7, 8, 9}, {10, 11, 12}}

    for (int row = 0; row < 2; row++)
    {
        for (int col = 0; col < 3; col++)
        {
            cout << myArray[row][col] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

### Output:



```
C:\Users\HP\OneDrive\Desktop\1.exe
7 8 9
10 11 12

-----
Process exited after 5.923 seconds with return value 0
Press any key to continue . . .
```

## Code 4:

```
#include <iostream>

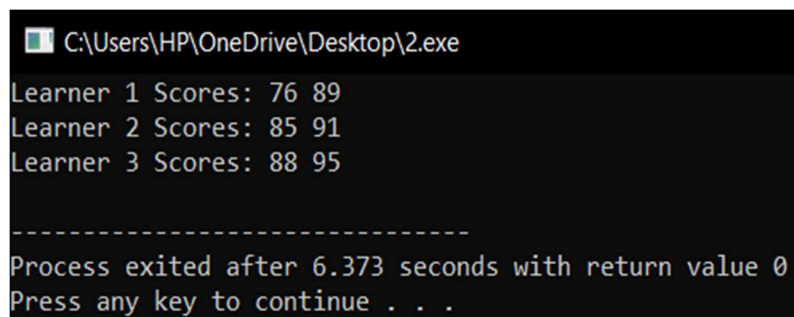
using namespace std;

int main()
{
    int scores[3][2] = {{76, 89},{85, 91},{88, 95}};

    for (int student = 0; student < 3; student++)
    {
        cout << "Learner " << student + 1 << " Scores: ";
        for (int subject = 0; subject < 2; subject++)
        {
            cout << scores[student][subject] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

## Output:



```
C:\Users\HP\OneDrive\Desktop\2.exe
Learner 1 Scores: 76 89
Learner 2 Scores: 85 91
Learner 3 Scores: 88 95

-----
Process exited after 6.373 seconds with return value 0
Press any key to continue . . .
```

## Lab 03

### Vectors

Vectors are part of the **Standard Template Library (STL)** and are a dynamic array-like structure that can grow and shrink in size as needed. Unlike regular arrays, vectors can dynamically resize, making them a flexible and efficient way to store collections of data.

#### **Syntax:**

```
#include <vector>
```

```
std::vector<type> vector_name;
```

---

#### **Code 1:**

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    vector<double> numbers;
```

```
    numbers.push_back(10.5);
```

```
    numbers.push_back(20.7);
```

```
    numbers.push_back(30.2);
```

```
    numbers.push_back(40.1);
```

```
    numbers.push_back(50.3);
```

```
    cout << "Initial Vector elements: ";
```

```
    for (int i = 0; i < numbers.size(); i++) {
```

```
        cout << numbers[i] << " ";
```

```
    }
```

```
    cout << endl;
```

```
    numbers.pop_back();
```

```
    cout << "After removing the last element, the vector contains: ";
```

---

```

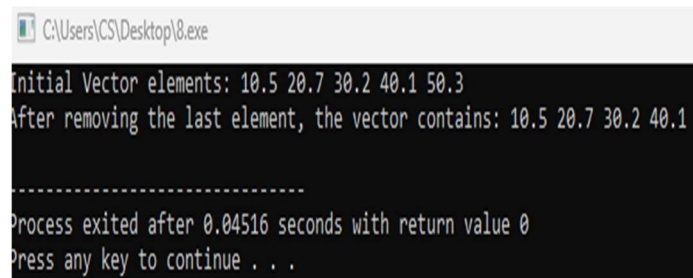
    for (int i = 0; i < numbers.size(); i++) {
        cout << numbers[i] << " ";
    }

    cout<<endl;

    return 0;
}

```

## Output:



```

C:\Users\CS\Desktop\8.exe
Initial Vector elements: 10.5 20.7 30.2 40.1 50.3
After removing the last element, the vector contains: 10.5 20.7 30.2 40.1
-----
Process exited after 0.04516 seconds with return value 0
Press any key to continue . . .

```

## Code 2:

```

#include <iostream>

#include <vector>

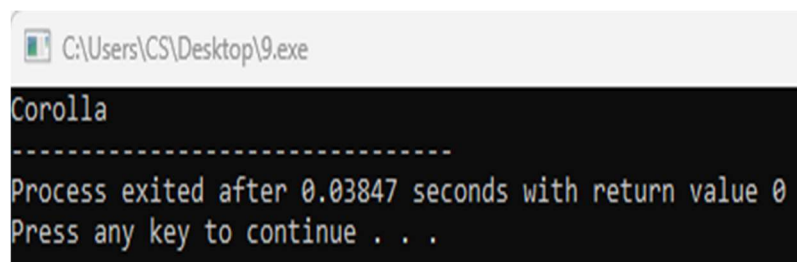
using namespace std;

int main() {
    vector<string> cars = {"Volvo", "BMW", "Ford", "Mazda"};
    cars[0] = "Corolla";
    cout << cars[0];

    return 0;}

```

## Output:



```

C:\Users\CS\Desktop\9.exe
Corolla
-----
Process exited after 0.03847 seconds with return value 0
Press any key to continue . . .

```

### Code 3:

```
#include <iostream>

#include <vector>

using namespace std;

int main() {

    vector<string> vehicleList = {"Civic", "City", "Ferarri", "Audi"}; // Changed 'cars' to 'vehicleList'

    vehicleList.push_back("Tesla"); // Changed 'cars' to 'vehicleList'

    for (string v : vehicleList) { // Changed 'car' to 'v'

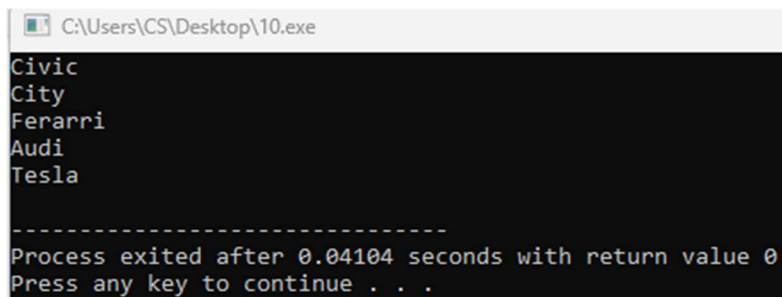
        cout << v << "\n"; // Changed 'car' to 'v'

    }

    return 0;

}
```

### Output:



```
C:\Users\CS\Desktop\10.exe
Civic
City
Ferarri
Audi
Tesla
-----
Process exited after 0.04104 seconds with return value 0
Press any key to continue . . .
```

### Code 4:

```
#include <iostream>

#include <vector>

using namespace std;

int main()
```

```

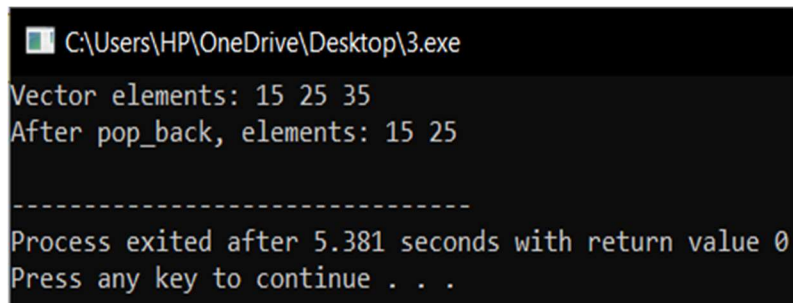
{
    vector<int> values;
    values.push_back(15);
    values.push_back(25);
    values.push_back(35);
    cout << "Vector elements: ";
    for (int index = 0; index < values.size(); index++)
    {
        cout << values[index] << " ";
    }
    cout << endl;
    values.pop_back();

    cout << "After pop_back, elements: ";
    for (int index = 0; index < values.size(); index++)
    {
        cout << values[index] << " ";
    }
    cout << endl;

    return 0;
}

```

## Output:



```

C:\Users\HP\OneDrive\Desktop\3.exe
Vector elements: 15 25 35
After pop_back, elements: 15 25

-----
Process exited after 5.381 seconds with return value 0
Press any key to continue . . .

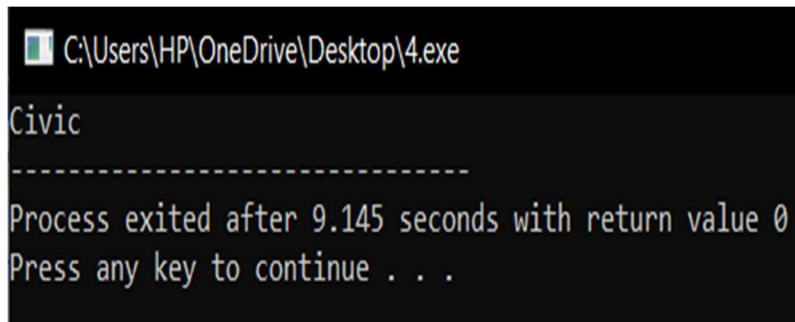
```

### Code 5:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<string> cars = {"Toyota", "Audi", "Honda", "Nissan"};
    cars[0] = "Civic";

    cout << cars[0];
    return 0;
}
```

### Output:



```
C:\Users\HP\OneDrive\Desktop\4.exe
Civic
-----
Process exited after 9.145 seconds with return value 0
Press any key to continue . . .
```

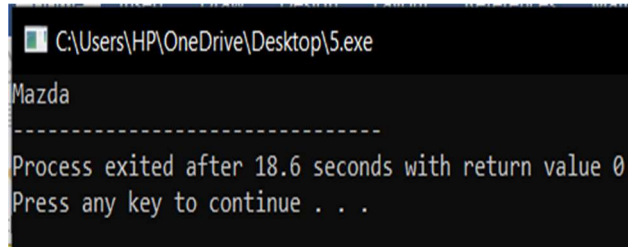
### Code 6:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<string> cars = {"Hyundai", "Chevrolet", "Kia", "Subaru"};
    cars.at(0) = "Mazda";
```



```
    cout << cars.at(0);  
    return 0;  
}
```

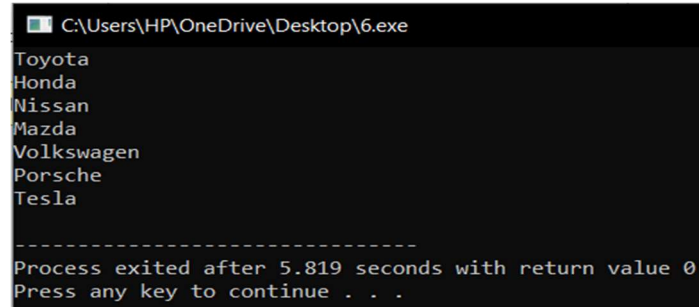
## Output:



## Code 7:

```
#include <iostream>  
#include <vector>  
using namespace std;  
int main()  
{  
    vector<string> cars = {"Toyota", "Honda", "Nissan", "Mazda", "Volkswagen", "Porsche"};  
    cars.push_back("Tesla");  
  
    for (string car : cars)  
    {  
        cout << car << "\n";  
    }  
    return 0;  
}
```

## Output:

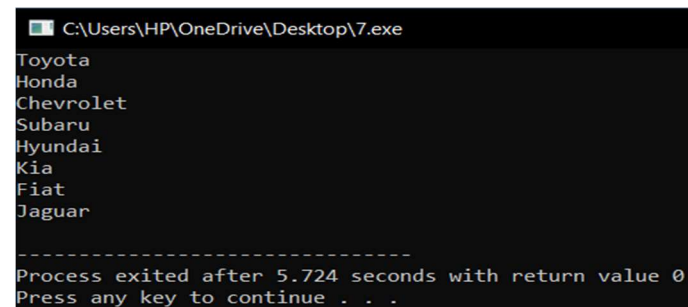


```
C:\Users\HP\OneDrive\Desktop\6.exe
Toyota
Honda
Nissan
Mazda
Volkswagen
Porsche
Tesla
-----
Process exited after 5.819 seconds with return value 0
Press any key to continue . . .
```

## Code 8:

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<string> cars = {"Toyota", "Honda", "Chevrolet", "Subaru"};
    cars.push_back("Hyundai");
    cars.push_back("Kia");
    cars.push_back("Fiat");
    cars.push_back("Jaguar");
    for (const string& car : cars) {
        cout << car << "\n"; }
    return 0;
}
```

## Output:



```
C:\Users\HP\OneDrive\Desktop\7.exe
Toyota
Honda
Chevrolet
Subaru
Hyundai
Kia
Fiat
Jaguar
-----
Process exited after 5.724 seconds with return value 0
Press any key to continue . . .
```

## Code 9:

```
#include <iostream>

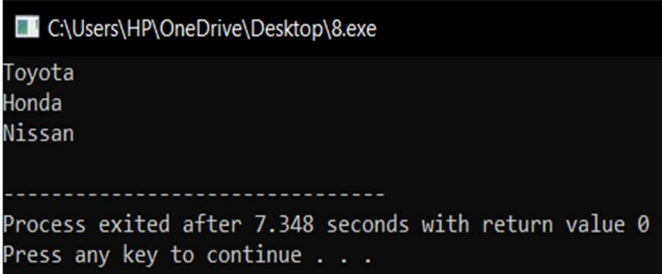
#include <vector>

using namespace std;

int main()
{
    vector<string> cars = {"Toyota", "Honda", "Nissan", "Hyundai"};

    cars.pop_back();
    for (string car : cars)
    {
        cout << car << "\n";
    }
    return 0;
}
```

## Output:



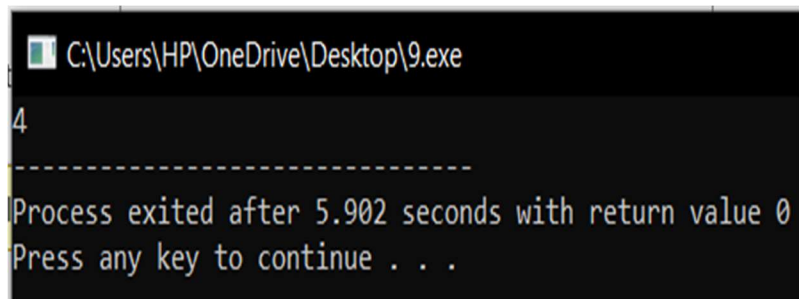
```
C:\Users\HP\OneDrive\Desktop\8.exe
Toyota
Honda
Nissan

-----
Process exited after 7.348 seconds with return value 0
Press any key to continue . . .
```

### Code 10:

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<string> cars = {"Toyota", "Audi", "Tesla", "Porsche"};
    cout << cars.size();
    return 0;
}
```

### Output:

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\HP\OneDrive\Desktop\9.exe". The command prompt displays the output "4" on the first line. Below it, a dashed line separates the output from the status message. The status message reads "Process exited after 5.902 seconds with return value 0" followed by "Press any key to continue . . .".

```
C:\Users\HP\OneDrive\Desktop\9.exe
4
-----
Process exited after 5.902 seconds with return value 0
Press any key to continue . . .
```

## LAB 04

### List

List is part of the **Standard Template Library (STL)**. It is a **doubly linked list** implementation that allows for efficient insertions and deletions from both ends of the list. Unlike vectors, lists do not allow random access to elements (i.e., you cannot directly access an element using an index), but they provide fast operations for adding and removing elements anywhere in the list.

### **Syntax:**

```
#include <list>

std::list<type> list_name;
```

---

### **Code 1:**

```
#include<iostream>

using namespace std;

int main(){

    string arr[5]={"oil" , "bread" , "eggs" , "vegetables" , "fruits"};

    for(int i=0 ; i<5 ; i++)

    {

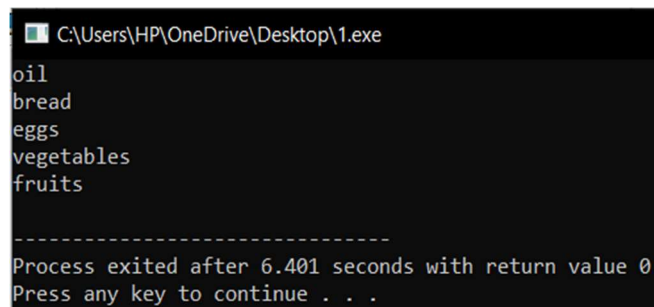
        cout<<arr[i]<<endl;

    }

    return 0;

}
```

### **Output:**



```
C:\Users\HP\OneDrive\Desktop\1.exe
oil
bread
eggs
vegetables
fruits
-----
Process exited after 6.401 seconds with return value 0
Press any key to continue . . .
```

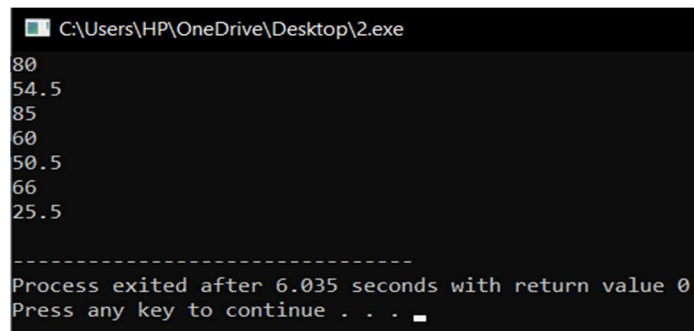
## Code 2:

```
#include<iostream>

using namespace std;

int main()
{
    float arr[7]={80 , 54.5 , 85 , 60 , 50.5 , 66 , 25.5 };
    for(int i=0 ; i<7 ; i++)
    {
        cout<<arr[i]<<endl;
    }
    return 0;
}
```

## Output:



```
C:\Users\HP\OneDrive\Desktop\2.exe
80
54.5
85
60
50.5
66
25.5
-----
Process exited after 6.035 seconds with return value 0
Press any key to continue . . .
```

## Code 3:

```
#include<iostream>

using namespace std;

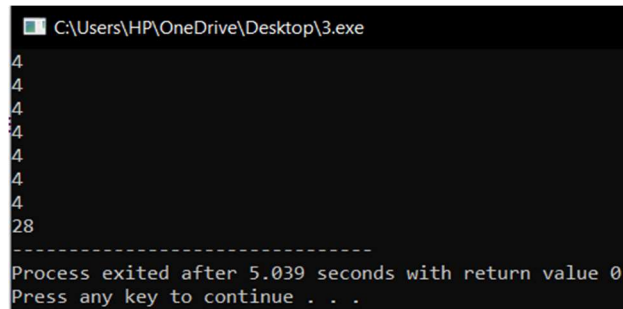
int main()
{
    float arr[7]={80 , 54.5 , 85 , 60 , 50.5 , 66 , 25.5 };
    for(int i=0 ; i<7 ; i++)
    {
```

```

        cout<<sizeof(arr[i])<<endl;
    }
    cout<<sizeof(arr);
    return 0;
}

```

## Output:



```

C:\Users\HP\OneDrive\Desktop\3.exe
4
4
4
4
4
4
4
28
-----
Process exited after 5.039 seconds with return value 0
Press any key to continue . . .

```

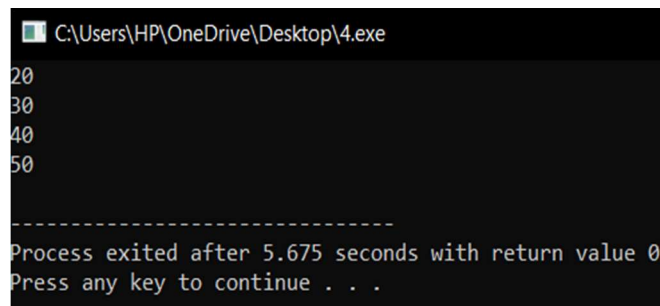
## Code 4:

```

#include<iostream>
#include<list>
using namespace std;
int main(){
    list<int> numbers = {20, 30, 40, 50};
    for(int num: numbers){
        cout<<num<<endl;
    }
    return 0;}

```

## Output:



```

C:\Users\HP\OneDrive\Desktop\4.exe
20
30
40
50
-----
Process exited after 5.675 seconds with return value 0
Press any key to continue . . .

```

## Code 5:

```
#include<iostream>

#include<list>

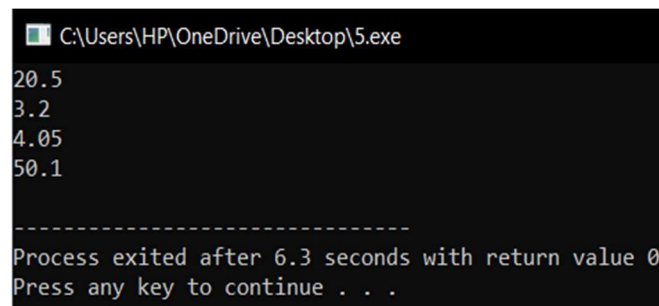
using namespace std;

int main()
{
    list<float> numbers = {20.5, 3.2, 4.05, 50.1};

    for(float num: numbers)
    {
        cout<<num<<endl;
    }

    return 0;
}
```

## Output:



```
C:\Users\HP\OneDrive\Desktop\5.exe
20.5
3.2
4.05
50.1

-----
Process exited after 6.3 seconds with return value 0
Press any key to continue . . .
```

## Code 6:

```
#include<iostream>

#include<list>

using namespace std;

int main()
{
    list<int> numbers = {1, 2, 3, 4, 5};

    numbers.reverse();
```

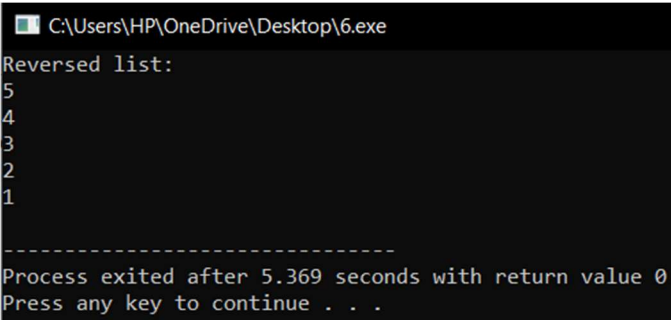


```

        cout << "Reversed list: " << endl;
    for (int num : numbers)
    {
        cout << num << " " << endl;
    }
    return 0;
}

```

## Output:



```

C:\Users\HP\OneDrive\Desktop\6.exe
Reversed list:
5
4
3
2
1
-----
Process exited after 5.369 seconds with return value 0
Press any key to continue . . .

```

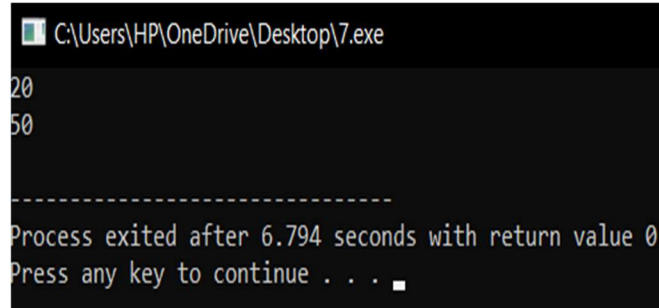
## Code 7:

```

#include<iostream>
#include<list>
using namespace std;
int main()
{
    list<int> numbers = {20, 30, 40, 50};
    cout<<numbers.front()<<endl;
    cout<<numbers.back()<<endl;
    return 0;
}

```

## Output:

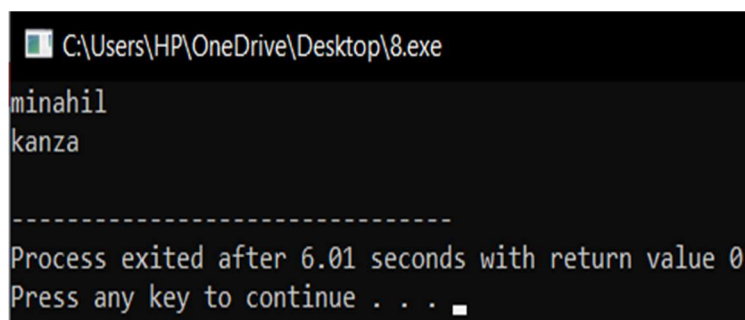


```
C:\Users\HP\OneDrive\Desktop\7.exe
20
50
-----
Process exited after 6.794 seconds with return value 0
Press any key to continue . . .
```

## Code 8:

```
#include<iostream>
#include<list>
using namespace std;
int main()
{
    list<string> names = {"humna", "yumna", "mishal", "ayesha"};
    names.front()="minahil";
    names.back() = "kanza";
    cout<<names.front()<<endl;
    cout<<names.back()<<endl;
    return 0;
}
```

## Output:



```
C:\Users\HP\OneDrive\Desktop\8.exe
minahil
kanza
-----
Process exited after 6.01 seconds with return value 0
Press any key to continue . . .
```

### Code 9:

```
#include<iostream>

#include<list>

using namespace std;

int main()
{
    list<string> names = {"humna", "yumna", "mishal", "ayesha"};

    names.push_front("minahil");

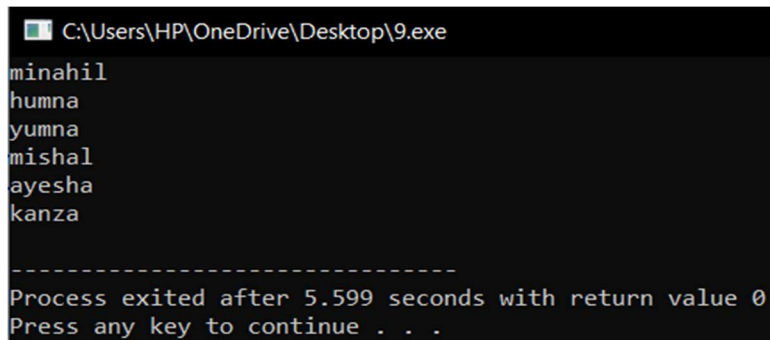
    names.push_back("kanza");

    for(string name : names)

        cout<<name<<endl;

    return 0;
}
```

### Output:



```
C:\Users\HP\OneDrive\Desktop\9.exe
minahil
humna
yumna
mishal
ayesha
kanza
-----
Process exited after 5.599 seconds with return value 0
Press any key to continue . . .
```

### Code 10:

```
#include<iostream>

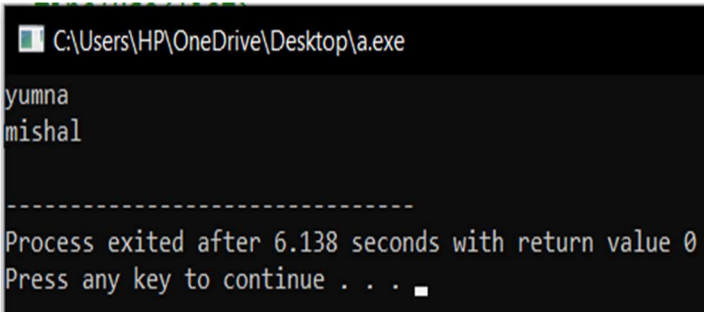
#include<list>

using namespace std;

int main()
{
```

```
list<string> names = {"humna", "yumna", "mishal", "ayesha"};
    names.pop_front();
    names.pop_back();
    for(string name : names)
        cout<<name<<endl;
    return 0;
}
```

## Output:



```
C:\Users\HP\OneDrive\Desktop\a.exe
yumna
mishal
-----
Process exited after 6.138 seconds with return value 0
Press any key to continue . . .
```

## **Lab 05**

### **Stack**

A stack in C++ is a linear data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed. Stacks are useful for tasks such as expression evaluation, backtracking, and managing function calls.

#### **Syntax:**

```
#include <stack>

stack<type> st;

stack<int> integer_stack;

stack<string> string_stack;
```

---

#### **Code 1:**

```
#include <iostream>

using namespace std;

#define n 100

class Stack {

    int* arr;

    int top;

public:

    Stack() {

        arr = new int[n];

        top = -1;

    }

    void push(int x) {

        if (top == n - 1) {
```

---

```

        cout << "Stack overflow" << endl;
        return;
    }
    arr[++top] = x;
}

void pop() {
    if (top == -1) {
        cout << "No elements to pop" << endl;
        return;
    }
    top--;
}

int top1() {
    if (top == -1) {
        cout << "Stack is empty" << endl;
        return -1;
    }
    return arr[top];
}

bool empty() {
    return top == -1;
}

~Stack() {
    delete[] arr;
}

```

```

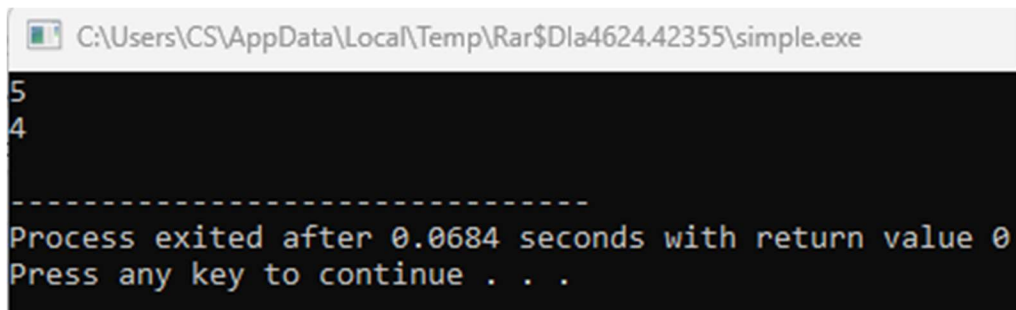
    }
};

int main() {
    Stack st;
    st.push(3);
    st.push(4);
    st.push(5);
    cout << st.top1() << endl;
    st.pop();
    cout << st.top1() << endl;

    return 0;
}

```

### Output:



```

5
4
-----
Process exited after 0.0684 seconds with return value 0
Press any key to continue . . .

```

### Code 2:

#### Infix to postfix (using library):

```

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

int prec(char c) {

```

```

        if (c == '^') {
            return 4;
        }
        else if (c == '*' || c == '/') {
            return 3;
        }
        else if (c == '-' || c == '+') {
            return 2;
        }
        else {
            return -1;
        }
    }

string infixTOpostfix(string s) {
    char stack[100];
    int top = -1;
    string res;

    for (int I = 0; I < s.length(); i++) {

        if ((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z')) {
            res += s[i];
        }

        else if (s[i] == '(' || s[i] == '{' || s[i] == '[') {
            stack[++top] = s[i];
        }
    }

```



```

else if (s[i] == ')' || s[i] == '}' || s[i] == ']') {
    while (top != -1 && stack[top] != '(') {
        res += stack[top--];
    }
    if (top != -1) {
        top--;
    }
}

else {
    while (top != -1 && prec(stack[top]) >= prec(s[i])) {
        res += stack[top--];
    }
    stack[++top] = s[i];
}

}

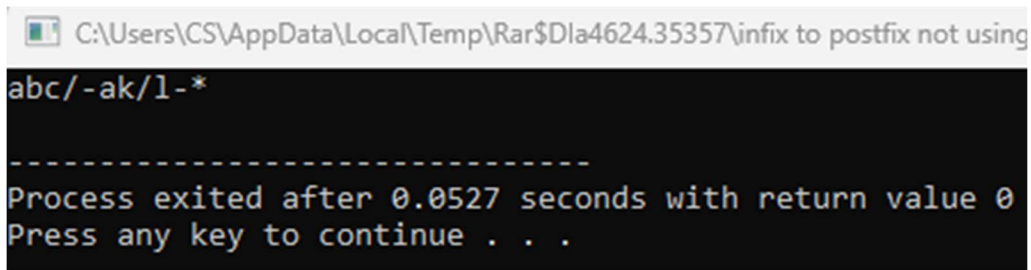
while (top != -1) {
    res += stack[top--];
}

return res;
}

int main() {
    cout << infixTOpostfix("(a-b/c)*(a/k-l)") << endl;
    return 0;
}

```

## Output:



```
C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.35357\infix to postfix not using
abc/-ak/l-*
-----
Process exited after 0.0527 seconds with return value 0
Press any key to continue . . .
```

## Code 3:

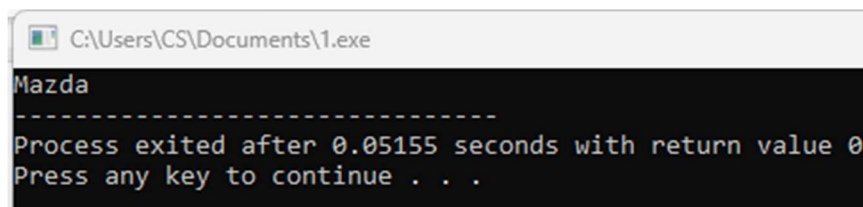
### Add top element:

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

```
int main() {
    stack<string> cars;

    cars.push("Volvo");
    cars.push("BMW");
    cars.push("Ford");
    cars.push("Mazda");
    cout << cars.top();
    return 0;
}
```

## Output:



```
C:\Users\CS\Documents\1.exe
Mazda
-----
Process exited after 0.05155 seconds with return value 0
Press any key to continue . . .
```

#### Code 4:

##### Change top:

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

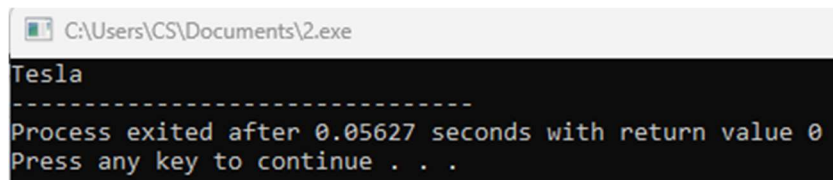
int main() {
    stack<string> cars;

    cars.push("Volvo");
    cars.push("BMW");
    cars.push("Ford");
    cars.push("Mazda");

    cars.top() = "Tesla";

    cout << cars.top();
    return 0;
}
```

##### Output:



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\CS\Documents\2.exe". The command prompt displays the output "Tesla" followed by a dashed line separator. Below the separator, it shows the message "Process exited after 0.05627 seconds with return value 0" and "Press any key to continue . . .".

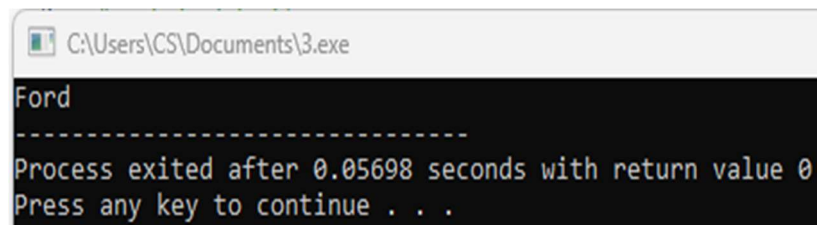
## Code 5:

### Removing elements:

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<string> cars;

    cars.push("Volvo");
    cars.push("BMW");
    cars.push("Ford");
    cars.push("Mazda");
    cars.pop();
    cout << cars.top();
    return 0;
}
```

### Output:



```
C:\Users\CS\Documents\3.exe
Ford
-----
Process exited after 0.05698 seconds with return value 0
Press any key to continue . . .
```

## **Lab 06**

### **Queue**

A queue stores multiple elements in a specific order, called **FIFO**. **FIFO** stands for **First in, First Out**. To visualize FIFO, think of a queue as people standing in line in a supermarket. The first person to stand in line is also the first who can pay and leave the supermarket. This way of organizing elements is called FIFO in computer science and programming.

#### **Syntax:**

```
#include <queue>

queue<type> q;

queue<int> integer_queue;

queue<string> string_queue;
```

---

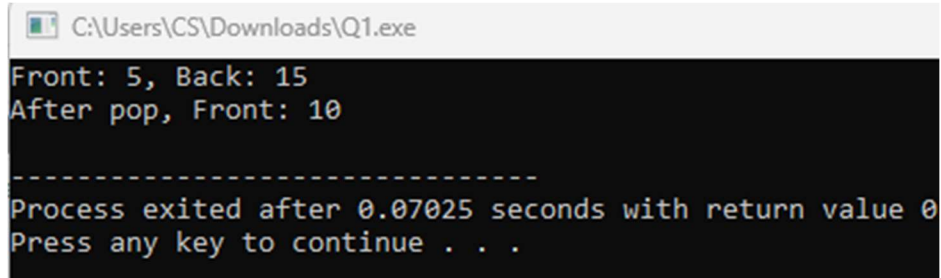
#### **Code 1:**

##### **Pop front, back:**

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

int main() {
    queue<int> q;
    q.push(5);
    q.push(10);
    q.push(15);
    cout << "Front: " << q.front() << ", Back: " << q.back() << endl;
    q.pop();
    cout << "After pop, Front: " << q.front() << endl;
    return 0;
}
```

## Output:



```
C:\Users\CS\Downloads\Q1.exe
Front: 5, Back: 15
After pop, Front: 10

-----
Process exited after 0.07025 seconds with return value 0
Press any key to continue . . .
```

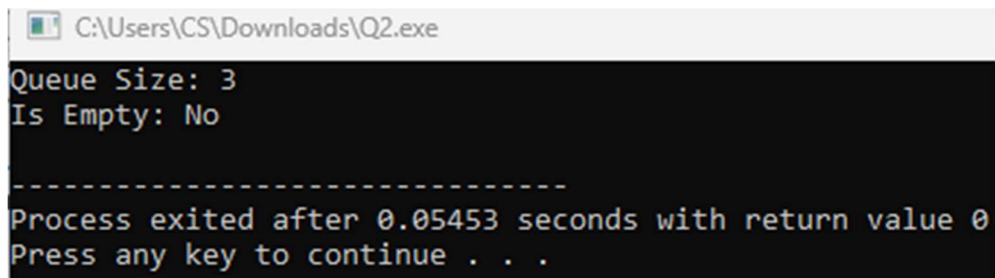
## Code 2:

### Queue size:

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

int main() {
    queue<int> q;
    q.push(1); q.push(2); q.push(3);
    cout << "Queue Size: " << q.size() << endl;
    cout << "Is Empty: " << (q.empty() ? "Yes" : "No") << endl;
    return 0;
}
```

## Output:



```
C:\Users\CS\Downloads\Q2.exe
Queue Size: 3
Is Empty: No

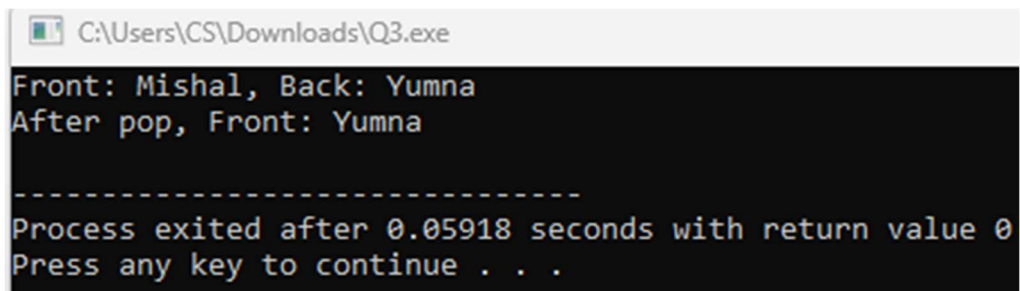
-----
Process exited after 0.05453 seconds with return value 0
Press any key to continue . . .
```

### Code 3:

#### Push elements:

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

#### Output:



```
C:\Users\CS\Downloads\Q3.exe
Front: Mishal, Back: Yumna
After pop, Front: Yumna
-----
Process exited after 0.05918 seconds with return value 0
Press any key to continue . . .
```

### Code 4:

#### Empty queue:

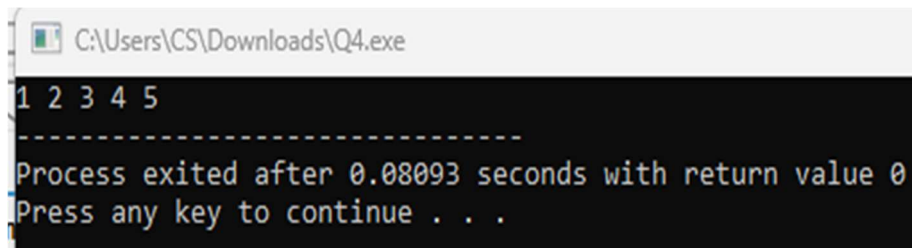
```
#include <iostream>
#include <queue>
using namespace std;
```

```

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

```

## Output:



```

C:\Users\CS\Downloads\Q4.exe
1 2 3 4 5
-----
Process exited after 0.08093 seconds with return value 0
Press any key to continue . . .

```

## Code 5:

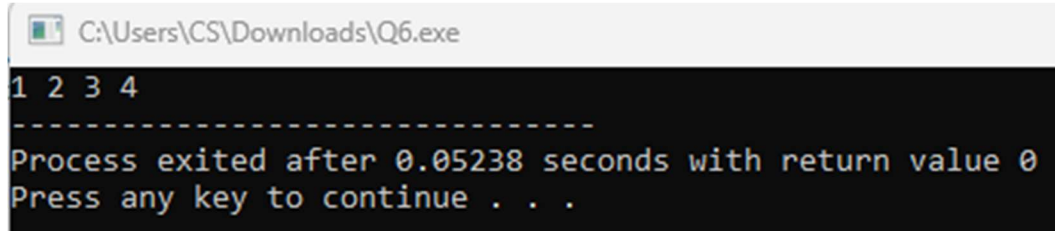
```

#include <iostream>
#include <queue>
using namespace std;
int main() {
    queue<int> q1, q2;
    q1.push(1); q1.push(2);
    q2.push(3); q2.push(4);
    while (!q1.empty()) { cout << q1.front() << " "; q1.pop(); }
    while (!q2.empty()) { cout << q2.front() << " "; q2.pop(); }
    return 0;
}

```



## Output:



```
C:\Users\CS\Downloads\Q6.exe  
1 2 3 4  
-----  
Process exited after 0.05238 seconds with return value 0  
Press any key to continue . . .
```

## **Lab 07**

### **Single Link list**

The singly linked list is the simplest form of linked list in which the node contain two members data and a next pointer that stores the address of the next node. Each node is a singly linked list is connected through the next pointer and the next pointer of the last node points to NULL denoting the end of the linked list.

#### **Syntax:**

```
struct node
{
    int data;
    struct node *next;
};
```

---

#### **Code**

##### **Linear search:**

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

class Node {
public:
    int val;
    Node* next;
    Node(int data) {
        val = data;
        next = NULL;
    }
};

void insert(Node*& head, int data) {
```

```

Node* newNode = new Node(data);
if (head == NULL) {
    head = newNode;
    return;
}
Node* temp = head;
while (temp->next != NULL) {
    temp = temp->next;
}
temp->next = newNode;
}

void search(Node* head, int v) {
    Node* temp = head;
    bool found = false;
    while (temp != NULL) {
        if (temp->val == v) {
            cout << "Value " << v << " found at node: " << temp << endl;
            found = true;
        }
        temp = temp->next;
    }
    if (!found) {
        cout << "Value " << v << " not found in the list." << endl;
    }
}

void displayList(Node* head) {

```

```

Node* temp = head;
while (temp != NULL) {
    cout << temp->val << " -> ";
    temp = temp->next;
}
cout << "NULL\n";
}

```

```

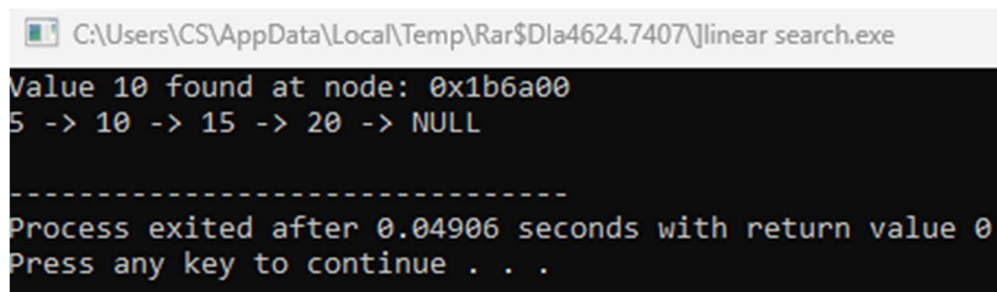
int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);

    search(head, 10);
    displayList(head);

    return 0;
}

```

## Output:



```

C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.7407\linear search.exe
Value 10 found at node: 0x1b6a00
5 -> 10 -> 15 -> 20 -> NULL

-----
Process exited after 0.04906 seconds with return value 0
Press any key to continue . . .

```

## Deletion

### **Code 1:**

#### **Delete tail:**

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

class Node {
public:
    int val;
    Node* next;
    Node(int data) {
        val = data;
        next = NULL;
    }
};

void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
```

```

}

void delTail(Node* head){
    Node* secondlast=head;
    while (secondlast->next->next != NULL){
        secondlast=secondlast->next;
    }
    Node* temp=secondlast->next;
    secondlast->next = NULL;
    delete temp;
}

```

```

void displayList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

```

```

int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);
    delTail(head);
}

```

```
displayList(head);  
}
```

## Output:



```
C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.8456\delelte tail.exe  
5 -> 10 -> 15 -> NULL  
-----  
Process exited after 0.07313 seconds with return value 0  
Press any key to continue . . .
```

## Code 2:

### Delete at position:

```
#include <iostream>  
using namespace std;  
  
class Node {  
public:  
    int val;  
    Node* next;  
    Node(int data) {  
        val = data;  
        next = NULL;  
    }  
};  
  
void insert(Node*& head, int data) {  
    Node* newNode = new Node(data);  
    if (head == NULL) {
```

```

        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void delatP(Node* &head, int pos){
    Node* prev=head;
    int currentpos=0;
    while (currentpos != pos-1){
        prev = prev->next;
        currentpos++;
    }
    Node* temp = prev->next;
    prev->next= prev->next->next;
    delete temp;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val << " -> ";
        temp = temp->next;
    }
}

```



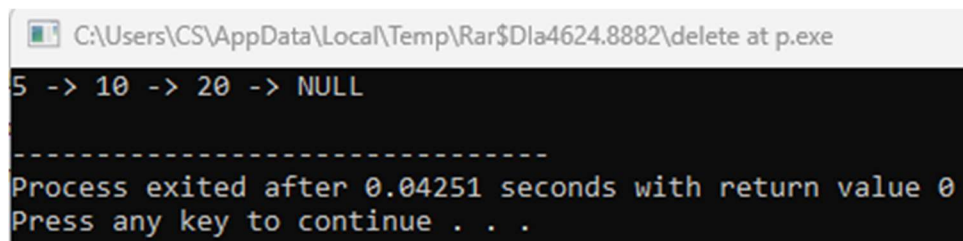
```

        cout << "NULL\n";
    }

int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);
    delatP(head,2);
    displayList(head);
}

```

## Output:



```

C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.8882\delete at p.exe
5 -> 10 -> 20 -> NULL
-----
Process exited after 0.04251 seconds with return value 0
Press any key to continue . . .

```

## Code 3:

### Delete at start:

```

#include <iostream>
using namespace std;
class Node {
public:
    int val;

```

```

Node* next;
Node(int data) {
    val = data;
    next = NULL;
}
};

void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void delatstart(Node*& head) {
    if (head == NULL) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}

void displayList(Node* head) {

```

```

Node* temp = head;
while (temp != NULL) {
    cout << temp->val << " -> ";
    temp = temp->next;
}
cout << "NULL\n";
}

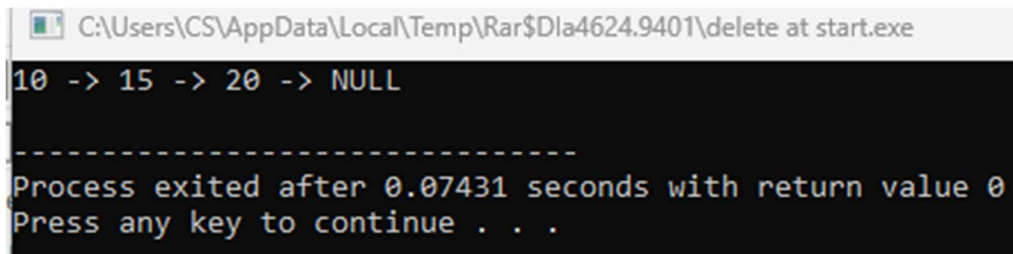
```

```

int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);
    delatstart(head);
    displayList(head);
}

```

## Output:



```

C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.9401\delete at start.exe
10 -> 15 -> 20 -> NULL
-----
Process exited after 0.07431 seconds with return value 0
Press any key to continue . . .

```

## Insertion

### **Code 1:**

#### **Insert at position:**

```
#include <iostream>

using namespace std;

class Node {
public:
    int val;
    Node* next;
    Node(int data) {
        val = data;
        next = NULL;
    }
};

void insert(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
```

```

}

void insertatstart(Node* &head, int data){
    Node* newnode = new Node(data);
    newnode->next = head;
    head = newnode;
}

void insertatP(Node* &head, int val, int pos){
    if(pos==0){
        insertatstart(head,val);
        return;
    }
    Node* newnode = new Node(val);
    Node* temp = head;
    int currentpos=0;
    while (currentpos!=pos-1){
        temp=temp->next;
        currentpos++;
    }
    newnode->next=temp->next;
    temp->next = newnode;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

```

```

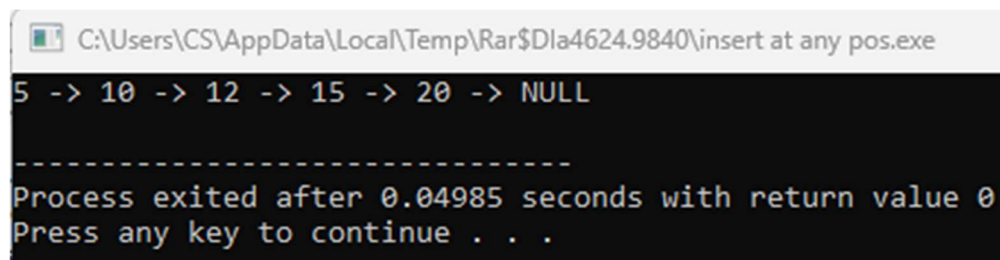
}

int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);
    insertatP(head,12,2);
    displayList(head);

    return 0;
}

```

## Output:



```

C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.9840\insert at any pos.exe
5 -> 10 -> 12 -> 15 -> 20 -> NULL
-----
Process exited after 0.04985 seconds with return value 0
Press any key to continue . . .

```

## Code 2:

### Insert at start:

```

#include <iostream>

using namespace std;

class Node {
public:
    int val;
    Node* next;

```

```

Node(int data) {
    val = data;
    next = NULL;
}
};

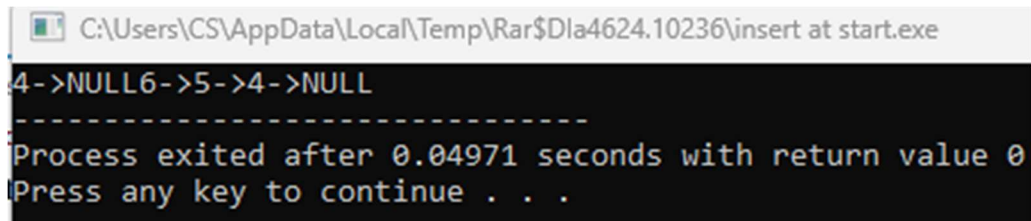
void insertatstart(Node* &head, int data){
    Node* newnode = new Node(data);
    newnode->next = head;
    head = newnode;
}

void display(Node* head){
    Node* temp = head;
    while (temp!=NULL){
        cout << temp->val << "->";
        temp=temp->next;
    }
    cout << "NULL";
}

int main (){
    Node* head = NULL;
    insertatstart(head,4);
    display(head);
    insertatstart(head,5);
    insertatstart(head,6);
    display(head);
}

```

**Output:**



```
C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.10236\insert at start.exe
4->NULL6->5->4->NULL
-----
Process exited after 0.04971 seconds with return value 0
Press any key to continue . . .
```

## Code 2:

### Insert at tail:

```
#include <iostream>
```

```
using namespace std;
```

```
class Node {
```

```
public:
```

```
    int val;
```

```
    Node* next;
```

```
    Node(int data) {
```

```
        val = data;
```

```
        next = NULL;
```

```
    }
```

```
};
```

```
void insert(Node*& head, int data) {
```

```
    Node* newNode = new Node(data);
```

```
    if (head == NULL) {
```

```
        head = newNode;
```

```
        return;
```

```
    }
```

```
    Node* temp = head;
```

```
    while (temp->next != NULL) {
```



```

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

void insertAtTail(Node* &head,int val){
    Node* newnode=new Node(val);
    Node* temp=head;
    while(temp->next!=NULL){
        temp = temp->next;
    }
    temp->next=newnode;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}

int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);

    displayList(head);
}

```

```

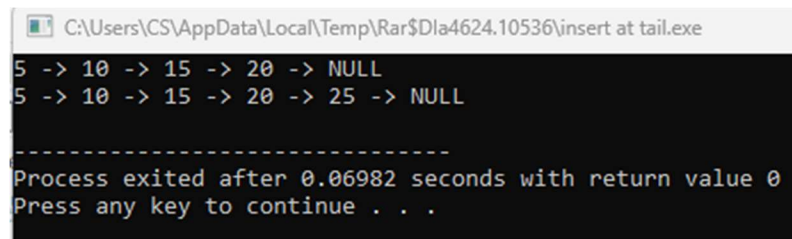
insertAtTail(head,25);

displayList(head);

return 0;
}

```

## Output:



```

C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.10536\insert at tail.exe
5 -> 10 -> 15 -> 20 -> NULL
5 -> 10 -> 15 -> 20 -> 25 -> NULL

-----
Process exited after 0.06982 seconds with return value 0
Press any key to continue . . .

```

## Code 3:

### Only insert:

```

#include <iostream>

using namespace std;

class Node {
public:
    int val;
    Node* next;
    Node(int data) {
        val = data;
        next = NULL;
    }
};

void insert(Node*& head, int data) {
    Node* newNode = new Node(data);

```

```

    if (head == NULL) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val << " -> ";
        temp = temp->next;
    }
    cout << "NULL\n";
}


int main() {
    Node* head = NULL;
    insert(head, 5);
    insert(head, 10);
    insert(head, 15);
    insert(head, 20);

    displayList(head);
    return 0;
}

```

}

## Output:



A screenshot of a Windows command prompt window. The title bar shows the file path: C:\Users\CS\AppData\Local\Temp\Rar\$Dla4624.10818\only insert .exe. The command prompt displays the output of a program: 5 -> 10 -> 15 -> 20 -> NULL. Below this, a dashed line separates the output from the program's exit message: Process exited after 0.05473 seconds with return value 0. The prompt then asks: Press any key to continue . . .

```
C:\Users\CS\AppData\Local\Temp\Rar$Dla4624.10818\only insert .exe  
5 -> 10 -> 15 -> 20 -> NULL  
-----  
Process exited after 0.05473 seconds with return value 0  
Press any key to continue . . .
```

## lab 08

### Circular link list

A **Circular Linked List** is a variation of a linked list in which all nodes are connected to form a circle.

---

### Insertion

#### **Code 1:**

#### **Inert at beginning:**

```
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
};

void insertBegin(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
```

```

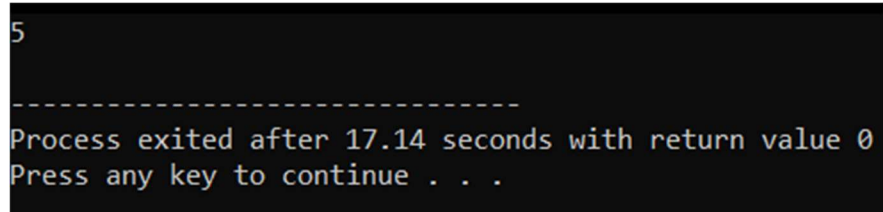
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = *head;
    *head = newNode;
}
}

void display(Node* head) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

int main() {
    Node* head = NULL;
    insertBegin(&head, 45);
    insertBegin(&head, 80);
    insertBegin(&head, 15);
    display(head);
    return 0;
}

```

## Output:



```
5
-----
Process exited after 17.14 seconds with return value 0
Press any key to continue . . .
```

## Code 2:

### Insert at beginning:

```
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
};

void insertMid(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else {
        Node* slow = *head;
        Node* fast = *head;
```

```

        while (fast->next != *head && fast->next->next != *head) {
            slow = slow->next;
            fast = fast->next->next;
        }
        newNode->next = slow->next;
        slow->next = newNode;
    }
}

```

```

void display(Node* head) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;}

int main() {
    Node* head = NULL;
    insertMid(&head, 10);
    insertMid(&head, 20);
    insertMid(&head, 30);
    insertMid(&head, 40);
    display(head);
    return 0;
}

```



```
}
```

## Output:

```
10 30 40 20
-----
Process exited after 15.31 seconds with return value 0
Press any key to continue . . .
```

## Code 3:

### Insert at end:

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

struct Node {
    int data;
    Node* next;
};

void insertEnd(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```

    }
    temp->next = newNode;
    newNode->next = *head;
}
}
void display(Node* head) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
int main() {
    Node* head = NULL;
    insertEnd(&head, 45);
    insertEnd(&head, 80);
    insertEnd(&head, 15);
    display(head);
    return 0;
}

```

### Output:

```
45 80 15
```

```
-----
Process exited after 15.83 seconds with return value 0
Press any key to continue . . .
```

## Deletion

### **Code 1:**

#### **Delete at start:**

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

struct Node {
    int data;
    Node* next;
};

void deleteStart(Node** head) {
    if (*head == NULL) {
        cout << "List is empty." << endl;
        return;
    }

    Node* temp = *head;

    if ((*head)->next == *head) {
        delete *head;
        *head = NULL;
        return;
    }

    Node* last = *head;
    while (last->next != *head) {
```

```

        last = last->next;
    }

    Node* newHead = (*head)->next;
    last->next = newHead;

    delete *head;
    *head = newHead;
}

void insertEnd(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
    }
}

```

```

void display(Node* head) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

```

```

int main() {
    Node* head = NULL;

    insertEnd(&head, 45);
    insertEnd(&head, 80);
    insertEnd(&head, 15);

    cout << "Original List: ";
    display(head);

    deleteStart(&head);
    cout << "After Deletion at Start: ";
    display(head);

    deleteStart(&head);
}

```

```

    cout << "After Deleting Again: ";
    display(head);

    deleteStart(&head);
    cout << "After Deleting All: ";
    display(head);

    return 0;
}

```

## Output:

```

Original List: 45 80 15
After Deletion at Start: 80 15
After Deleting Again: 15
After Deleting All: List is empty.

-----
Process exited after 13.84 seconds with return value 0
Press any key to continue . . .

```

## Code 2:

### Delete at mid:

```

#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
};

void deleteMid(Node** head) {
    if (*head == NULL) {

```

```

        cout << "List is empty." << endl;
        return;
    }

    if ((*head)->next == *head) {
        delete *head;
        *head = NULL;
        return;
    }

    Node* slow = *head;
    Node* fast = *head;
    Node* prev = NULL;

    while (fast != *head && fast->next != *head) {
        prev = slow;
        slow = slow->next;
        fast = fast->next->next;
    }

    if (prev != NULL) {
        prev->next = slow->next;
        if (slow == *head) {
            *head = slow->next;
        }
        delete slow;
    }
}

```

```

void insertEnd(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
    }
}

```

```

void display(Node* head) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

```



```
    } while (temp != head);  
    cout << endl;  
}  
  
int main() {  
    Node* head = NULL;  
  
    insertEnd(&head, 45);  
    insertEnd(&head, 80);  
    insertEnd(&head, 15);  
    insertEnd(&head, 60);  
    insertEnd(&head, 25);  
  
    cout << "Original List: ";  
    display(head);  
  
    deleteMid(&head);  
    cout << "After Deletion at Mid: ";  
    display(head);  
  
    deleteMid(&head);  
    cout << "After Deleting Again: ";  
    display(head);  
  
    deleteMid(&head);  
    cout << "After Deleting All: ";  
    display(head);
```

```
    return 0;
}
```

## Output:

```
Original List: 45 80 15 60 25
After Deletion at Mid: 45 80 15 60 25
After Deleting Again: 45 80 15 60 25
After Deleting All: 45 80 15 60 25

-----
Process exited after 15.84 seconds with return value 0
Press any key to continue . . .
```

## Code 3:

### Delete at end:

```
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
};

void deleteEnd(Node** head) {
    if (*head == NULL) {
        cout << "List is empty." << endl;
        return;
    }

    if ((*head)->next == *head) {
        delete *head;
```

```
    *head = NULL;
    return;
}
```

```
Node* temp = *head;
Node* prev = NULL;
```

```
while (temp->next != *head) {
    prev = temp;
    temp = temp->next;
}
```

```
prev->next = *head;
delete temp;
}
```

```
void insertEnd(Node** head, int data) {
```

```
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;
```

```
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
```

```
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
```

```

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

void display(Node* head) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

int main() {
    Node* head = NULL;

    insertEnd(&head, 45);
    insertEnd(&head, 80);
    insertEnd(&head, 15);
    insertEnd(&head, 60);
    insertEnd(&head, 25);
}

```

```
cout << "Original List: ";  
display(head);  
  
deleteEnd(&head);  
cout << "After Deletion at End: ";  
display(head);  
  
deleteEnd(&head);  
cout << "After Deleting Again: ";  
display(head);  
  
deleteEnd(&head);  
cout << "After Deleting All: ";  
display(head);  
  
return 0;  
}
```

## Output:

```
Original List: 45 80 15 60 25  
After Deletion at End: 45 80 15 60  
After Deleting Again: 45 80 15  
After Deleting All: 45 80  
  
-----  
Process exited after 20.69 seconds with return value 0  
Press any key to continue . . .
```

## Lab 9

### Doubly link list

A doubly linked list is a type of linked list where each node contains three parts:

- Data – Stores the value.
- Next Pointer – Points to the next node.
- Previous Pointer – Points to the previous node.

---

### Insertion

#### **Code 1:**

#### **Insert at start:**

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

class Node {
public:

    int val;
    Node* next;
    Node* prev;
    Node(int data)
    {
        val=data;
        next=NULL;
        prev=NULL;
    }
};
```

```

class DOUBLELINKLIST{
public:
Node* head;
Node* tail;
DOUBLELINKLIST(){
head = NULL;
tail = NULL;
}
void insertHead(int val){
Node* new_node = new Node(val);
if (head==NULL){
head=new_node;
tail= new_node;
return;
}

new_node->next = head;
head->prev=new_node;
head=new_node;
return;
}

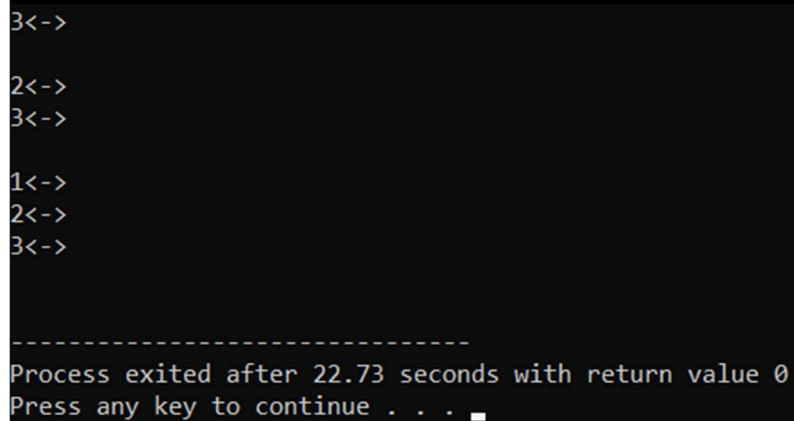
void display(){
Node* temp=head;
while (temp!=NULL){
cout << temp->val << "<->"<<endl;
temp=temp->next;
}
}

```

```
cout << endl;
}
};

int main ()
{
    DOUBLELINKLIST dll;
    dll.insertHead(3);
    dll.display();
    dll.insertHead(2);
    dll.display();
    dll.insertHead(1);
    dll.display();
}
```

## Output:



```
3<->
2<->
3<->
1<->
2<->
3<->

-----
Process exited after 22.73 seconds with return value 0
Press any key to continue . . . █
```



## Code 2:

### Insert at any position:

```
#include <iostream>

using namespace std;

class Node {
public:

    int val;
    Node* next;
    Node* prev;
    Node(int data){
        val=data;
        next=NULL;
        prev=NULL;
    }
};

class DOUBLELINKLIST{
public:

    Node* head;
    Node* tail;
    DOUBLELINKLIST(){
        head = NULL;
        tail = NULL;
    }
}
```

```

    void insertend(int val){
        Node* new_node = new Node(val);
        if (tail==NULL){
            head=new_node;
            tail= new_node;
            return;
        }
        new_node->prev = tail;
        tail->next=new_node;
        tail=new_node;
        return;
    }

    void display(){
        Node* temp=head;
        while (temp!=NULL){
            cout << temp->val << "<->"<<endl;
            temp=temp->next;
        }
        cout << endl;
    }

    void insertatP(int val, int k){
        int count=0;
        Node* temp = head;
        while (count <(k-1)){
            temp = temp->next;
            count++;
        }
        Node* new_node = new Node(val);

```

```

new_node->next = temp->next;
temp->next=new_node;
new_node->prev=temp;
new_node->next->prev=new_node;
return;
}
};
int main (){

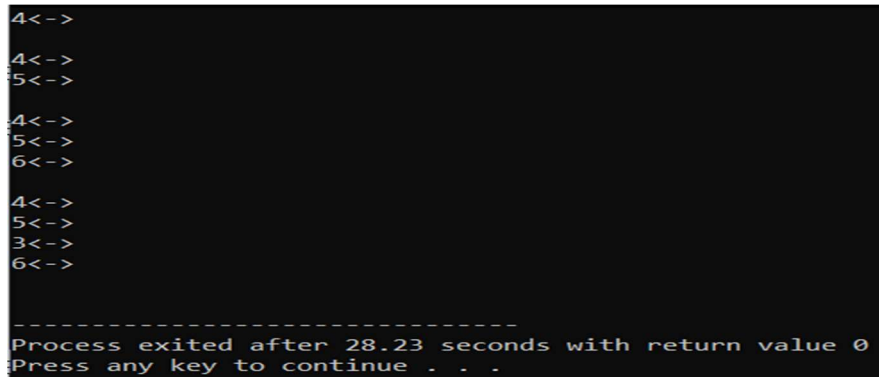
```

```

DOUBLELINKLIST dll;
dll.insertend(4);
dll.display();
dll.insertend(5);
dll.display();
dll.insertend(6);
dll.display();
dll.insertatP(3,2);
dll.display();
return 0;
}

```

## Output:



```

4<->
4<->
5<->
4<->
5<->
6<->
4<->
5<->
6<->
4<->
5<->
3<->
6<->
-----
Process exited after 28.23 seconds with return value 0
Press any key to continue . . .

```

## Deletion

### **Code 1:**

#### **Delete at start:**

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

class Node {
public:
    int val;
    Node* next;
    Node* prev;

    Node(int data) {
        val = data;
        next = NULL;
        prev = NULL;
    }
};

class DOUBLELINKLIST {
public:
    Node* head;
    Node* tail;

    DOUBLELINKLIST() {
        head = NULL;
        tail = NULL;
    }
};
```

```

}

void insert(int val) {
    Node* new_node = new Node(val);
    if (head == NULL) {
        head = new_node;
        tail = new_node;
    } else {
        tail->next = new_node;
        new_node->prev = tail;
        tail = new_node;
    }
}

```

```

void deleteAThead() {
    if (head == NULL) {
        return;
    }
    Node* temp = head;
    head = head->next;
    if (head == NULL) {
        tail = NULL;
    } else {
        head->prev = NULL;
    }
    delete temp;
}

```

```

void display() {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val;
        if (temp->next != NULL) {
            cout << " <-> ";
        }
        temp = temp->next;
    }
    cout << endl;
}
};

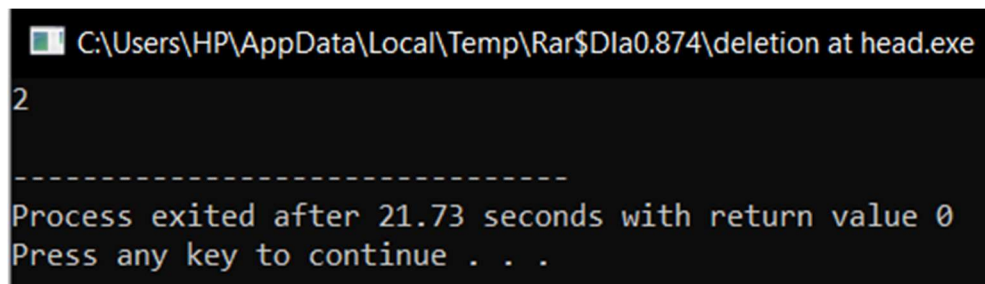
```

```

int main() {
    DOUBLELINKLIST dll;
    dll.insert(3);
    dll.insert(2);
    dll.deleteAThead();
    dll.display();
    return 0;
}

```

## Output:



```

C:\Users\HP\AppData\Local\Temp\Rar$Dla0.874\deletion at head.exe
2
-----
Process exited after 21.73 seconds with return value 0
Press any key to continue . . .

```

## Code 2:

### Delete at any position:

```
#include <iostream>

using namespace std;

class Node {
public:
    int val;
    Node* next;
    Node* prev;

    Node(int data) {
        val = data;
        next = NULL;
        prev = NULL;
    }
};

class DOUBLELINKLIST {
public:
    Node* head;
    Node* tail;

    DOUBLELINKLIST() {
        head = NULL;
        tail = NULL;
    }
};
```

```

void insert(int val) {
    Node* new_node = new Node(val);
    if (head == NULL) {
        head = new_node;
        tail = new_node;
    } else {
        tail->next = new_node;
        new_node->prev = tail;
        tail = new_node;
    }
}

void del(int p) {
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }

    Node* temp = head;
    int count = 1;

    while (temp != NULL && count < p) {
        temp = temp->next;
        count++;
    }

    if (temp == NULL) {
        cout << "Position out of bounds." << endl;
    }
}

```



```

        return;
    }

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

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

    if (temp == head)
        head = temp->next;

    if (temp == tail)
        tail = temp->prev;

    delete temp;
}

void display() {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val;
        if (temp->next != NULL) {
            cout << " <-> ";
        }
        temp = temp->next;
    }
    cout << endl;
}

```

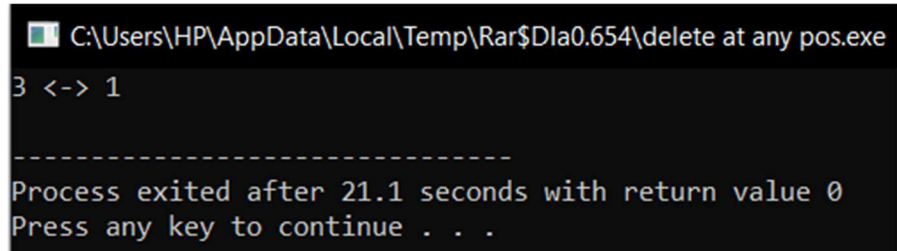
```

    }
};

int main() {
    DOUBLELINKLIST dll;
    dll.insert(3);
    dll.insert(2);
    dll.insert(1);
    dll.del(2);
    dll.display();
    return 0;
}

```

## Output:



```

C:\Users\HP\AppData\Local\Temp\Rar$Dla0.654\delete at any pos.exe
3 <-> 1
-----
Process exited after 21.1 seconds with return value 0
Press any key to continue . . .

```

## Code 3:

### Delete at end:

```

#include <iostream>
using namespace std;

class Node {
public:
    int val;

```

```

Node* next;
Node* prev;

Node(int data) {
    val = data;
    next = NULL;
    prev = NULL;
}
};

class DOUBLELINKLIST {
public:
    Node* head;
    Node* tail;

    DOUBLELINKLIST() {
        head = NULL;
        tail = NULL;
    }

    void insert(int val) {
        Node* new_node = new Node(val);
        if (head == NULL) {
            head = new_node;
            tail = new_node;
        } else {
            tail->next = new_node;
            new_node->prev = tail;

```

```
        tail = new _node;
    }
}
```

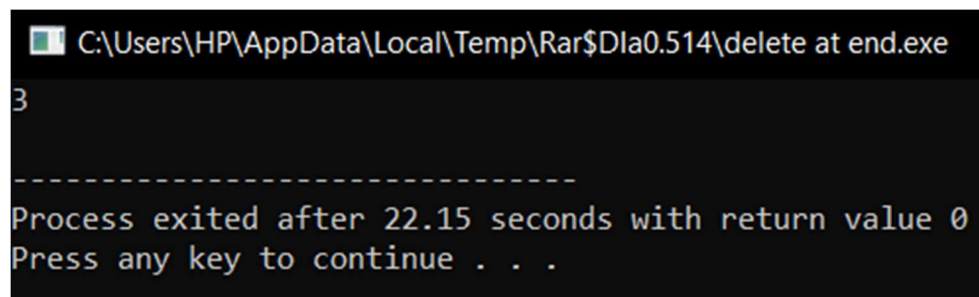
```
void del(){
    if (head==NULL){
        return;
    }
    Node* temp = tail;
    tail = tail->prev;
    if(head==NULL){
        tail=NULL;
    }
    else{
        tail->next = NULL;
    }
    delete temp;
}
```

```
void display() {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->val;
        if (temp->next != NULL) {
            cout << " <-> ";
        }
        temp = temp->next;
    }
}
```

```
        cout << endl;
    }
};

int main() {
    DOUBLELINKLIST dll;
    dll.insert(3);
    dll.insert(2);
    dll.del();
    dll.display();
    return 0;
}
```

## Output:



```
C:\Users\HP\AppData\Local\Temp\Rar$Dla0.514\delete at end.exe
3
-----
Process exited after 22.15 seconds with return value 0
Press any key to continue . . .
```

## **Lab 10**

### **Binary Search Tree**

A **Binary Search Tree** is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree. This property is called the BST property and every binary search tree follows this property as it allows efficient insertion, deletion, and search operations in a tree.

---

#### **Code 1:**

##### **Insertion:**

```
#include<iostream>

using namespace std;

struct parentNode{

    int val;

    parentNode* LC;

    parentNode* RC;

    parentNode(int data){

        val = data;

        LC = NULL;

        RC = NULL;

    }

};

parentNode* newNode(parentNode* Root , int val){

    if(Root == NULL)

        return new parentNode(val);
```

```

if(Root->val == val)
return Root;

if(val > Root->val)
Root->RC = newNode(Root->RC, val);
else
Root->LC = newNode(Root->LC, val);
return Root;
}

void inorderTraversal(parentNode* Root) {
if (Root == NULL)
return;
inorderTraversal(Root->LC);
cout << Root->val << " ";
inorderTraversal(Root->RC);
}

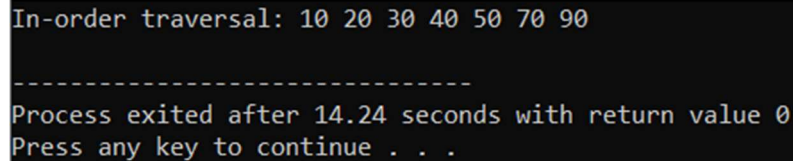
int main() {
parentNode* root = NULL;

root = newNode(root, 10);
root = newNode(root, 20);
root = newNode(root, 40);
root = newNode(root, 50);
root = newNode(root, 30);
root = newNode(root, 70);
root = newNode(root, 90);
cout << "In-order traversal: ";

```

```
inorderTraversal(root);  
cout << endl;  
return 0;  
}
```

## Output:



```
In-order traversal: 10 20 30 40 50 70 90  
-----  
Process exited after 14.24 seconds with return value 0  
Press any key to continue . . .
```

## Code 2:

### Searching:

```
#include <iostream>  
using namespace std;  
struct parentNode {  
    int data;  
    parentNode* LC;  
    parentNode* RC;  
  
    parentNode(int val) {  
        data = val;  
        LC = NULL;  
        RC = NULL;  
    }  
};  
  
parentNode* search(parentNode* Root, int data) {  
    if (Root == NULL || Root->data == data)  
        return Root;
```



```

if (Root->data < data)
return search(Root->RC, data);

return search(Root->LC, data);
}

int main() {

parentNode* Root = new parentNode(80);
Root->LC = new parentNode(20);
Root->RC = new parentNode(40);
Root->LC->LC = new parentNode(30);
Root->LC->RC = new parentNode(10);
Root->RC->LC = new parentNode(70);
Root->RC->RC = new parentNode(50);

int value;

cout << "Enter value to search: ";
cin >> value;

if (search(Root, value) != NULL)
cout << "Found" << endl;
else
cout << "Not Found" << endl;
return 0;
}

```

**Output:**

```
Enter value to search: 10
Not Found

-----
Process exited after 74.75 seconds with return value 0
Press any key to continue . . .
```

### Code 3:

#### Duplicate:

```
#include <iostream>

using namespace std;

struct parentNode {
    int data;
    parentNode* LC;
    parentNode* RC;

    parentNode(int val) {
        data = val;
        LC = nullptr;
        RC = nullptr;
    }
};

void duplicate(parentNode* Root, int val) {

    if (Root == nullptr) {
        cout << "Duplicate not found" << endl;
        return;
    }
```

```

if (Root->data == val) {
    cout << "Duplicate found" << endl;
    return;
}

if (val < Root->data) {
    duplicate(Root->LC, val);
}

else {
    duplicate(Root->RC, val);
}
}

int main() {

    parentNode* Root = new parentNode(50);
    Root->LC = new parentNode(30);
    Root->RC = new parentNode(70);
    Root->LC->LC = new parentNode(20);
    Root->LC->RC = new parentNode(40);
    Root->RC->LC = new parentNode(60);
    Root->RC->RC = new parentNode(80);
    int val = 40;
    duplicate(Root, val);
    return 0;
}

```

## Output:

```
Duplicate found
-----
Process exited after 15.78 seconds with return value 0
Press any key to continue . . .
```

## Code 4:

### Deletion:

```
#include <iostream>
using namespace std;
struct parentNode {
    int val;
    parentNode* LC;
    parentNode* RC;
};

parentNode* createNode(int val) {
    parentNode* newNode = new parentNode();
    newNode->val = val;
    newNode->LC = nullptr;
    newNode->RC = nullptr;
    return newNode;
}

void inOrder(parentNode* Root) {
    if (Root != nullptr) {
        inOrder(Root->LC);
        cout << Root->val << " ";
    }
}
```

```
inOrder(Root->RC);  
}  
}
```

```
parentNode* inOrderPredecessor(parentNode* root) {  
    root = root->LC;  
    while (root->RC != nullptr) {  
        root = root->RC;  
    }  
    return root;  
}
```

```
parentNode* deleteNode(parentNode* Root, int val) {  
    if (Root == nullptr)  
        return nullptr;
```

```
    if (val < Root->val) {  
        Root->LC = deleteNode(Root->LC, val);  
    } else if (val > Root->val) {  
        Root->RC = deleteNode(Root->RC, val);  
    } else {
```

```
        if (Root->LC == nullptr && Root->RC == nullptr) {  
            delete Root;  
            return nullptr;  
        } else if (Root->LC == nullptr) {  
            parentNode* temp = Root->RC;  
            delete Root;
```

```

return temp;
} else if (Root->RC == nullptr) {
parentNode* temp = Root->LC;
delete Root;
return temp;
} else {

parentNode* temp = inOrderPredecessor(Root);
Root->val = temp->val;
Root->LC = deleteNode(Root->LC, temp->val);
}
}
return Root;
}

int main() {

parentNode* Root = createNode(12);
parentNode* n1 = createNode(7);
parentNode* n2 = createNode(15);
parentNode* n3 = createNode(5);
parentNode* n4 = createNode(9);
parentNode* n5 = createNode(8);


Root->LC = n1;
Root->RC = n2;
n1->LC = n3;
n1->RC = n4;
n4->LC = n5;

```

```
cout << "Before Deletion:" << endl;
```

```
inOrder(Root);
```

```
cout << "\n";
```

```
Root = deleteNode(Root, 9);
```

```
cout << "After Deletion:" << endl;
```

```
inOrder(Root);
```

```
cout << "\n";
```

```
delete n5;
```

```
delete n4;
```

```
delete n3;
```

```
delete n2;
```

```
delete n1;
```

```
delete Root;
```

```
return 0;
```

```
}
```

## Output:

```
Before Deletion:
5 7 8 9 12 15
After Deletion:
5 7 8 12 15

-----
Process exited after 13.85 seconds with return value 0
Press any key to continue . . .
```

## Code 5:

### Traversal (in-order, pre-order, post-order):

```
#include <iostream>

using namespace std;

struct parentNode {
    int data;
    parentNode* LC;
    parentNode* RC; };

parentNode* createNode(int data) {
    parentNode* n = new parentNode();
    n->data = data;
    n->LC = nullptr;
    n->RC = nullptr;
    return n;
}

void preOrder(parentNode* Root) {
    if (Root != nullptr) {
        cout << Root->data << " ";
        preOrder(Root->LC);
        preOrder(Root->RC);
    }
}

void postOrder(parentNode* Root) {
    if (Root != nullptr) {
        postOrder(Root->LC);
        postOrder(Root->RC);
    }
}
```



```
cout << Root->data << " "; }}
```

```
void inOrder(parentNode* Root) {  
if (Root != nullptr) {  
cout << Root->data << " ";  
inOrder(Root->RC);  
}}
```

```
int main() {
```

```
parentNode* n = createNode(1);  
parentNode* n1 = createNode(2);  
parentNode* n2 = createNode(3);  
parentNode* n3 = createNode(4);  
parentNode* n4 = createNode(5);  
parentNode* n5 = createNode(6);
```

```
n->LC = n1;  
n->RC = n2;  
n1->LC = n3;  
n1->RC = n4;  
n2->RC = n5;
```

```
cout << "Inorder Traversal:" << endl;  
inOrder(n);  
cout << "\n";
```

```
cout << "Preorder Traversal:" << endl;  
preOrder(n);
```

```
cout << "\n";

cout << "Postorder Traversal:" << endl;
postOrder(n);
cout << "\n";

delete n4;
delete n3;
delete n2;
delete n1;
delete n;
return 0;
}
```

## Output:

```
Inorder Traversal:
4 2 5 1 3 6
Preorder Traversal:
1 2 4 5 3 6
Postorder Traversal:
4 5 2 6 3 1

-----
Process exited after 13.51 seconds with return value 0
Press any key to continue . . .
```