The University of Faisalabad

# Data structure

# Lab manual

Submitted to: Mam Irsha Qureshi

Submitted by: Taibah Shahbaz

Registration # 2023-BSAI-024

Semester: 3

# LAB NO 1

Introduction to c++ and reviewing websites such as w3school also memorizing syntax of basic c++

# LAB NO 2

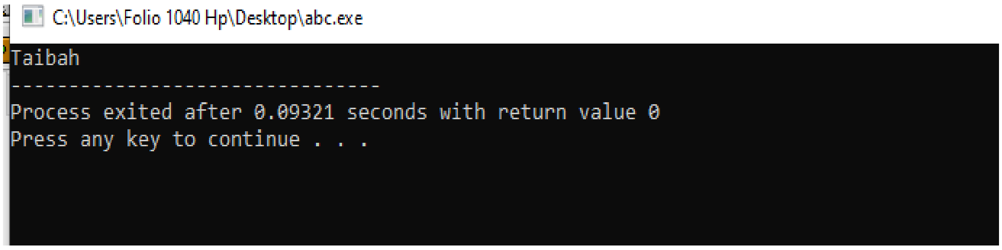
## What is Array?

An array is a data structure that stores a fixed-size sequential collection of elements of the same type. In other words, it’s a collection of variables (called elements), all of the same type, stored under a single variable name.

## PROGRAM NO 1:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string names[4] = {"Taibah", "Ahmed", "Fiza", "Maira"};
cout << names[0];
return 0;
}
```

## OUTPUT:



C:\Users\Folio 1040 Hp\Desktop\abc.exe

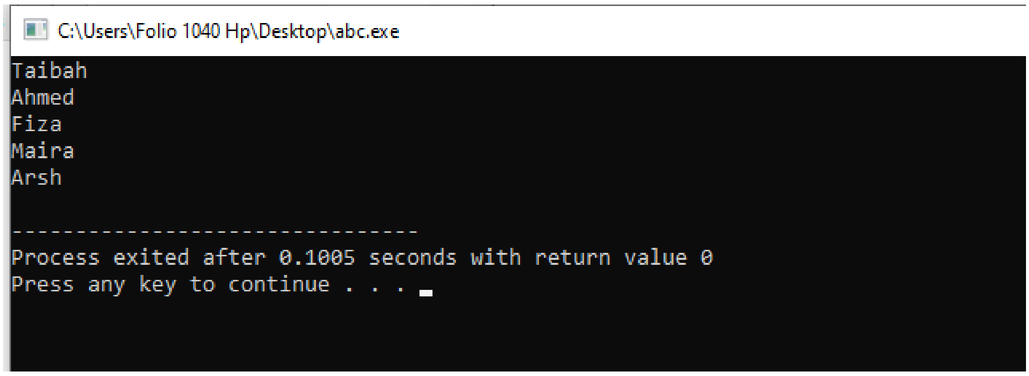
```
Taibah
--------------------------------
Process exited after 0.09321 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string names[5] = {"Taibah", "Ahmed", "Fiza", "Maira", "Arsh"};
for (int i = 0; i < 5; i++) {
```

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

# OUTPUT:

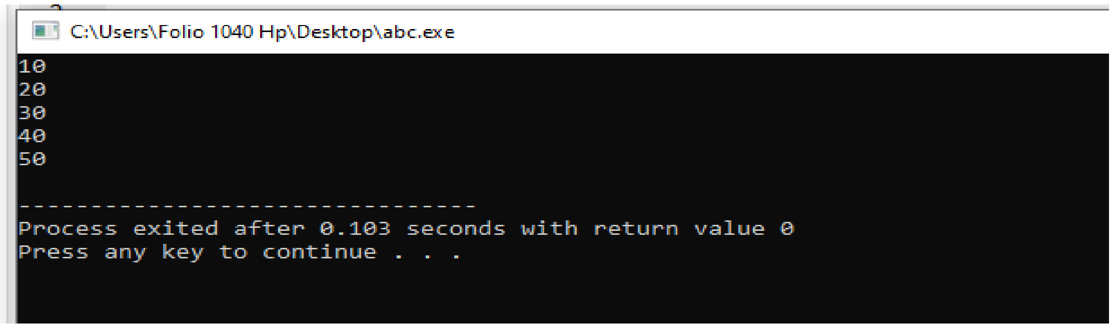


```
Taibah
Ahmed
Fiza
Maira
Arsh

--------------------------------
Process exited after 0.1005 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3:

```
#include <iostream>
using namespace std;
int main() {
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i = 0; i < 5; i++) {
cout << myNumbers[i] << "\n";
}
return 0;
}
```

# OUTPUT:

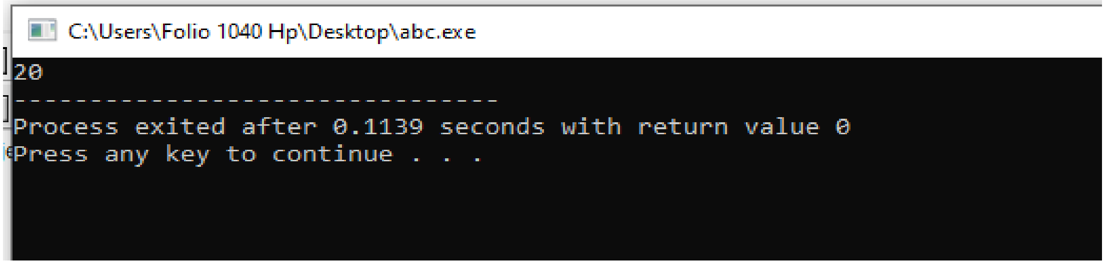


```
10
20
30
40
50

--------------------------------
Process exited after 0.103 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4:

```
#include <iostream>
using namespace std;
int main() {
int myNumbers[5] = {10, 20, 30, 40, 50};
cout << sizeof(myNumbers);
return 0;
}
```

# OUTPUT:

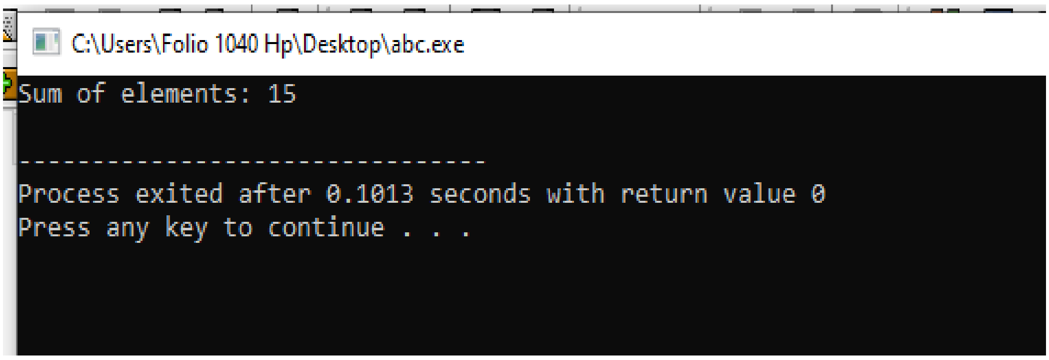


```
20
--------------------------------
Process exited after 0.1139 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 5:

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

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int sum = 0;

    for (int i = 0; i < 5; i++) {
        sum += arr[i];
    }
    cout << "Sum of elements: " << sum << endl;
    return 0;
}
```

## OUTPUT:

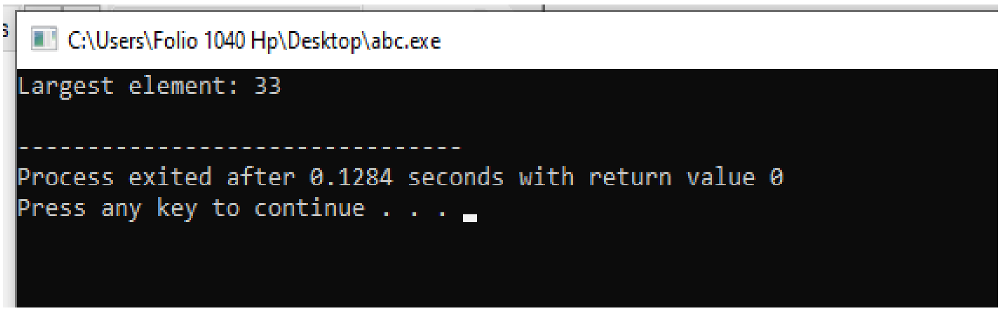


```
Sum of elements: 15

--------------------------------
Process exited after 0.1013 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 6:

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

int main() {
    int arr[5] = {10, 25, 7, 33, 15};
    int max = arr[0];

    for (int i = 1; i < 5; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    cout << "Largest element: " << max << endl;
    return 0;
}
```

# OUTPUT:

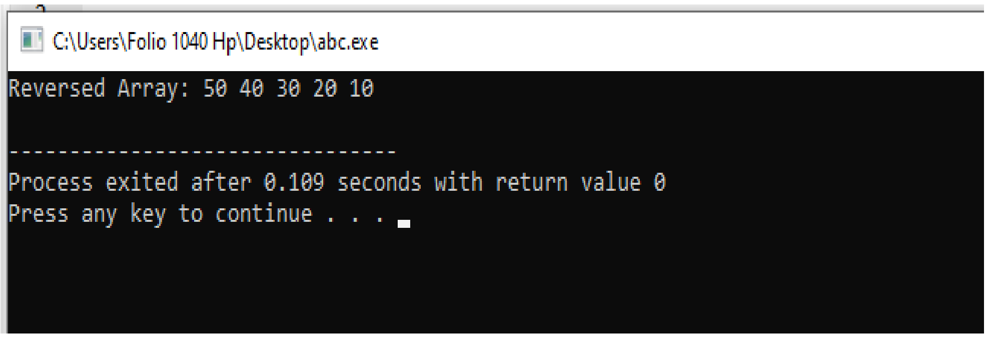


```
Largest element: 33

--------------------------------
Process exited after 0.1284 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 7:

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

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    cout << "Reversed Array: ";

    for (int i = 4; i >= 0; i--) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

# OUTPUT:

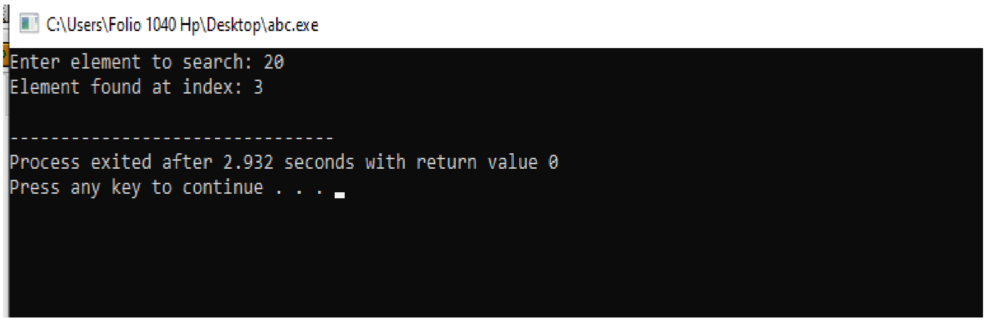


```
Reversed Array: 50 40 30 20 10

--------------------------------
Process exited after 0.109 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 8:

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

int main() {
    int arr[5] = {5, 8, 12, 20, 25};
    int search, found = -1;

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

    for (int i = 0; i < 5; i++) {
        if (arr[i] == search) {
            found = i;
            break;
        }
    }
    if (found != -1)
        cout << "Element found at index: " << found << endl;
    else
        cout << "Element not found" << endl;
    return 0;
}
```

## OUTPUT:

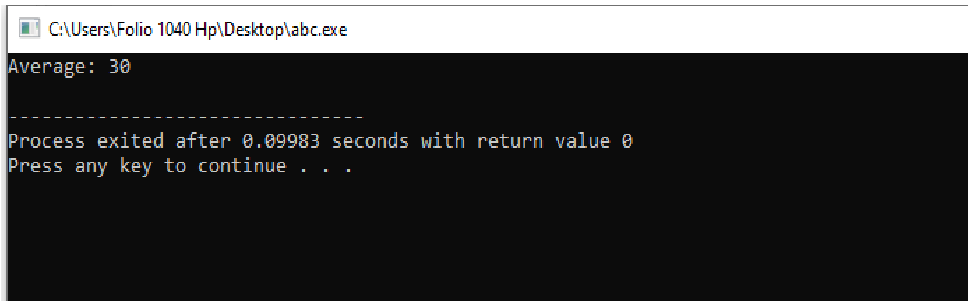


```
Enter element to search: 20
Element found at index: 3

--------------------------------
Process exited after 2.932 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 9:

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

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int sum = 0;
    float average;

    for (int i = 0; i < 5; i++) {
        sum += arr[i];
    }

    average = sum / 5.0;
    cout << "Average: " << average << endl;
    return 0;
}
```

## OUTPUT:



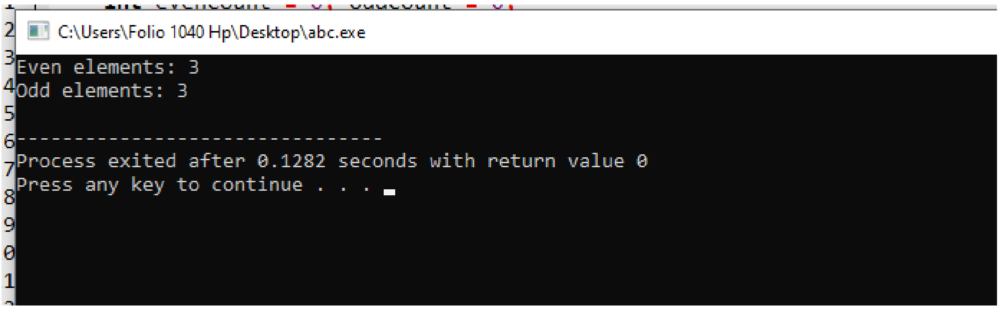
```
Average: 30

--------------------------------
Process exited after 0.09983 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 10:

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

```
int main() {
    int arr[6] = {1, 2, 3, 4, 5, 6};
    int evenCount = 0, oddCount = 0;

    for (int i = 0; i < 6; i++) {
        if (arr[i] % 2 == 0)
            evenCount++;
        else
            oddCount++;
    }

    cout << "Even elements: " << evenCount << endl;
    cout << "Odd elements: " << oddCount << endl;
    return 0;
}
```

## OUTPUT:



```
Even elements: 3
Odd elements: 3

--------------------------------
Process exited after 0.1282 seconds with return value 0
Press any key to continue . . .
```

# LAB NO 3

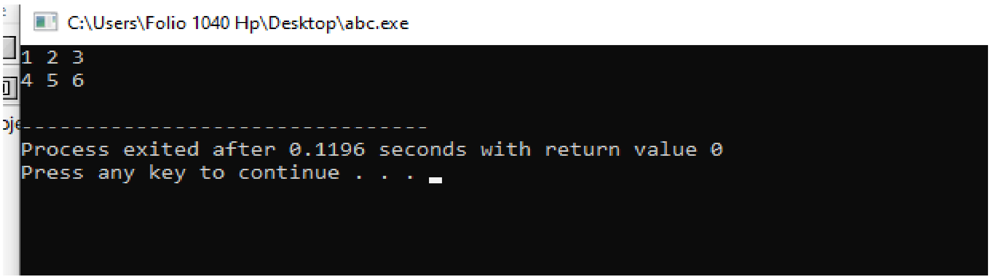
## Multidimensional array

A **multidimensional array** is an array of arrays, where each element is itself an array. In a **2D array**, elements are arranged in rows and columns, forming a matrix-like structure. This allows you to store data in a tabular form, making it ideal for scenarios like storing matrices, tables, or grids.

## PROGRAM NO 1:

```
#include <iostream>
using namespace std;
int main() {
    // Declare and initialize a 2x3 array (2 rows, 3 columns)
    int matrix[2][3] = {
        {1, 2, 3}, // First row
        {4, 5, 6}  // Second row
    };
    // Print the 2D array
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";
// Access elements using row and column indices
        }
        cout << endl;  // Newline after each row
    }
    return 0;
}
```

## OUTPUT:



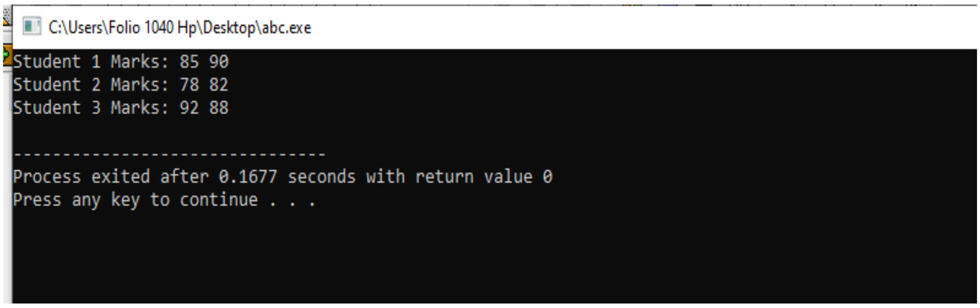
C:\Users\Folio 1040 Hp\Desktop\abc.exe

```
1 2 3
4 5 6

--------------------------------
Process exited after 0.1196 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2:

```
#include <iostream>
using namespace std;
int main() {
    // Declare and initialize a 3x2 array to store marks for 3 students in 2 subjects
    int marks[3][2] = {
        {85, 90},  // Marks for Student 1 in Subject 1 and Subject 2
        {78, 82},  // Marks for Student 2 in Subject 1 and Subject 2
        {92, 88}   // Marks for Student 3 in Subject 1 and Subject 2
    };
    // Display the marks
    for (int i = 0; i < 3; i++) {
        cout << "Student " << i+1 << " Marks: ";
        for (int j = 0; j < 2; j++) {
            cout << marks[i][j] << " ";  // Print each student's marks
        }
        cout << endl;
    }
    return 0;
}
```

## Output:

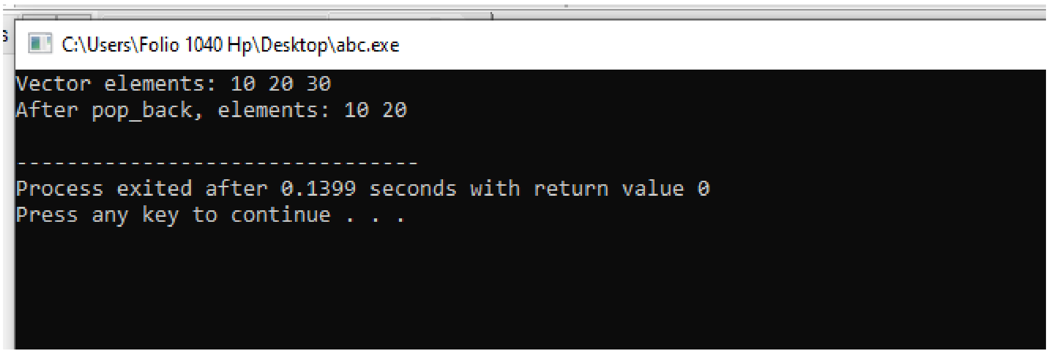


```
Student 1 Marks: 85 90
Student 2 Marks: 78 82
Student 3 Marks: 92 88

--------------------------------
Process exited after 0.1677 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3:

```
#include <iostream>
#include <vector> // Include the vector header
using namespace std;
int main() {
    // Create a vector to store integers
    vector<int> numbers;
    // Add elements to the vector
    numbers.push_back(10); // Add 10
    numbers.push_back(20); // Add 20
    numbers.push_back(30); // Add 30
    // Display the elements of the vector
    cout << "Vector elements: ";
    for (int i = 0; i < numbers.size(); i++) { // Use size() to get the number of elements
        cout << numbers[i] << " "; // Access elements using the index
    }
    cout << endl;
    // Remove the last element
    numbers.pop_back(); // Removes 30
    // Display the updated vector
    cout << "After pop_back, elements: ";
    for (int i = 0; i < numbers.size(); i++) {
        cout << numbers[i] << " "; // Print updated vector
    }
    cout << endl;
    return 0;
}
```

# OUTPUT:



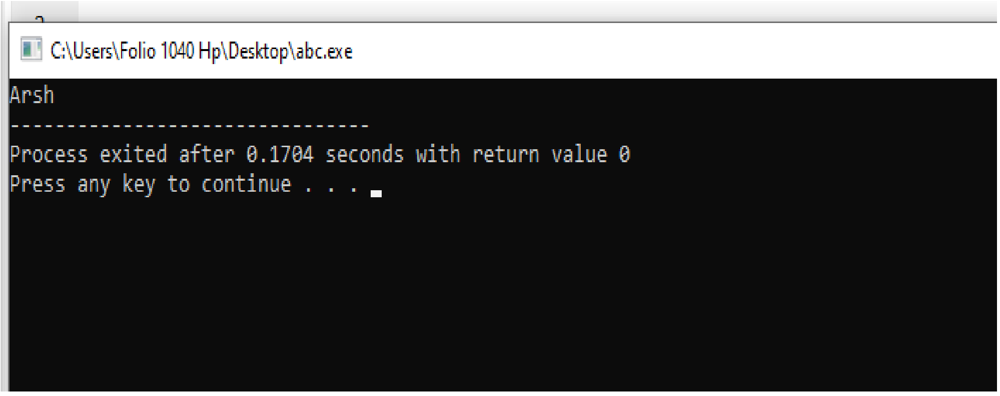
```
Vector elements: 10 20 30
After pop_back, elements: 10 20

--------------------------------
Process exited after 0.1399 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4:

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

```
int main() {
  vector<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};

  // Change the value of the first element
  names[0] = "Arsh";

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

## OUTPUT:

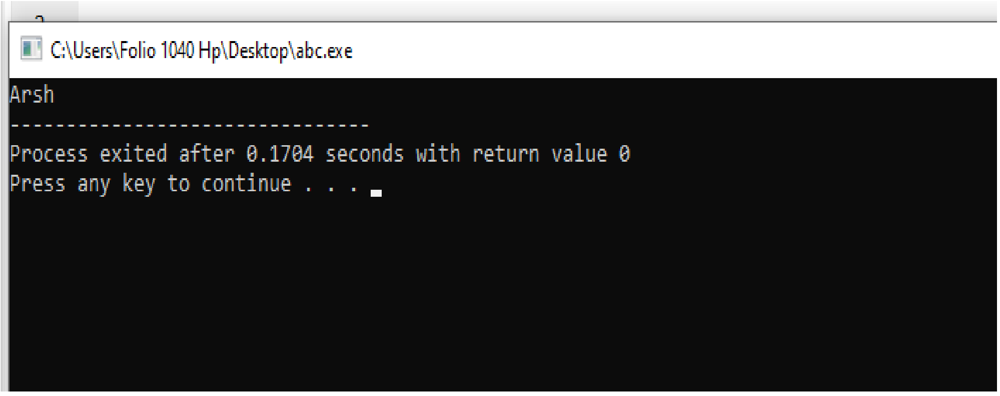


```
Arsh
--------------------------------
Process exited after 0.1704 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 5:

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

int main() {
  vector<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};

  // Change the value of the first element
  names.at(0) = "Arsh";

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

## OUTPUT:

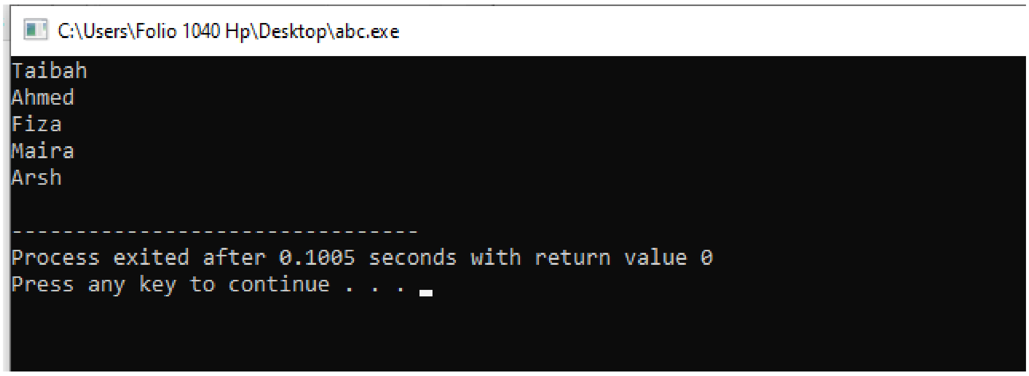


```
Arsh
--------------------------------
Process exited after 0.1704 seconds with return value 0
Press any key to continue . . .
```

**PROGRAM NO 6:**

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};
  names.push_back("Arsh");
  for (string name : names) {
    cout << name << "\n";
  }
  return 0;
}
```

**OUTPUT:**



```
Taibah
Ahmed
Fiza
Maira
Arsh

--------------------------------
Process exited after 0.1005 seconds with return value 0
Press any key to continue . . .
```

# LAB NO 4

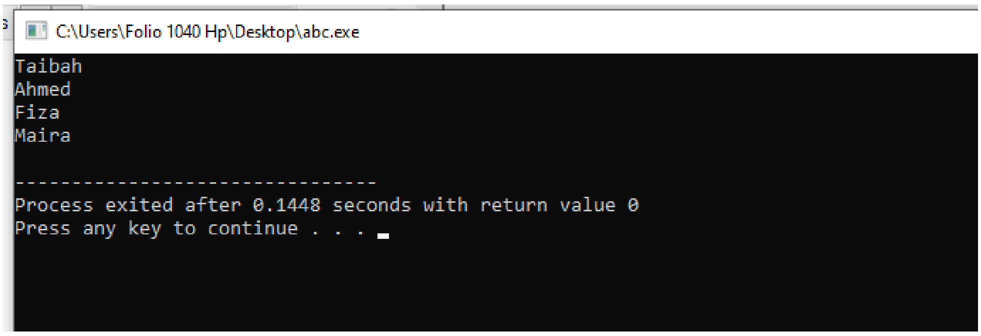
## VECTOR

A list is similar to a vector in that it can store multiple elements of the same type and dynamically grow in size.

## PROGRAM NO 1:

```
#include <iostream>
#include <list>
using namespace std;
int main() {
  // Create a list called cars that will store strings
  list<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};

  // Print list elements
  for (string name: names) {
    cout << name << "\n";
  }
  return 0;
}
```

## OUTPUT:



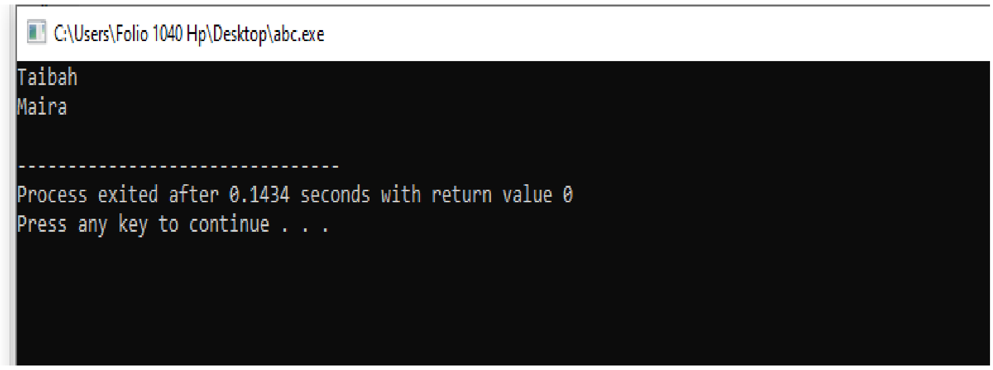
C:\Users\Folio 1040 Hp\Desktop\abc.exe

```
Taibah
Ahmed
Fiza
Maira

--------------------------------
Process exited after 0.1448 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2:

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

int main() {
  // Create a list called cars that will store strings
  list<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};

  // Get the first element
  cout << names.front() << "\n";

  // Get the last element
  cout << names.back() << "\n";
  return 0;
}
```

# OUTPUT:

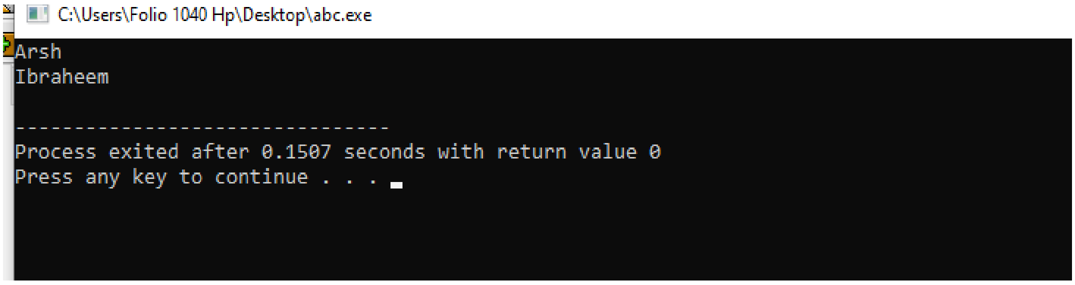


```
Taibah
Maira

--------------------------------
Process exited after 0.1434 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3:

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

int main() {
  list<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};

  // Change the value of the first element
  names.front() = “Arsh ";

  // Change the value of the last element
  names.back() = "Ibraheem";

  cout << names.front() << "\n";
  cout << names.back() << "\n";
  return 0;
}
```

# OUTPUT:

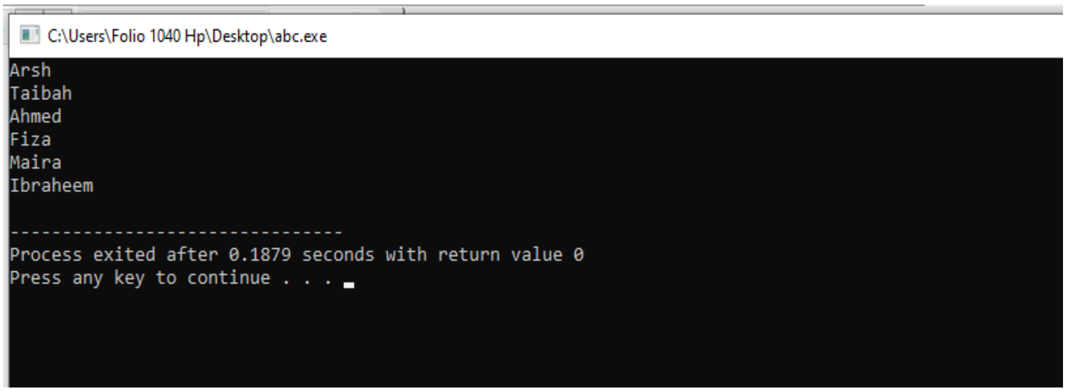


```
Arsh
Ibraheem

--------------------------------
Process exited after 0.1507 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4:

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

int main() {
  list<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};
  // Add an element at the beginning
  names.push_front("Arsh");

  // Add an element at the end
  names.push_back("Ibraheem");

  // Print list elements
  for (string name : names) {
    cout << name << "\n";
  }

  return 0;
}
```

**OUTPUT:**

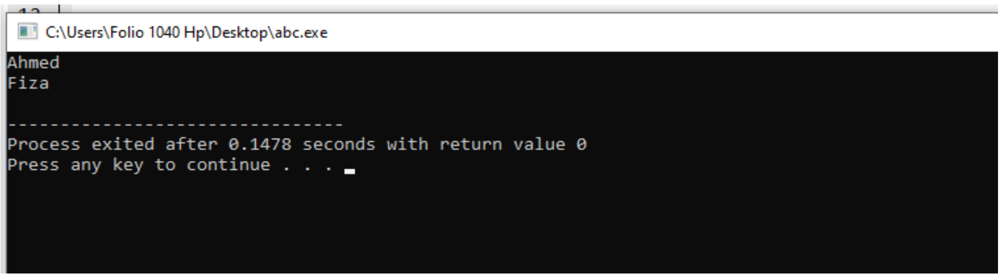


```
Arsh
Taibah
Ahmed
Fiza
Maira
Ibraheem

--------------------------------
Process exited after 0.1879 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 5:

```
#include <iostream>
#include <list>
using namespace std;
int main() {
  list<string> names = {"Taibah", "Ahmed", "Fiza", "Maira"};

  // Remove the first element
  names.pop_front();

  // Remove the last element
  names.pop_back();

  // Print list elements
  for (string name : names) {
    cout << name << "\n";
  }

  return 0;
}
```

**OUTPUT:**

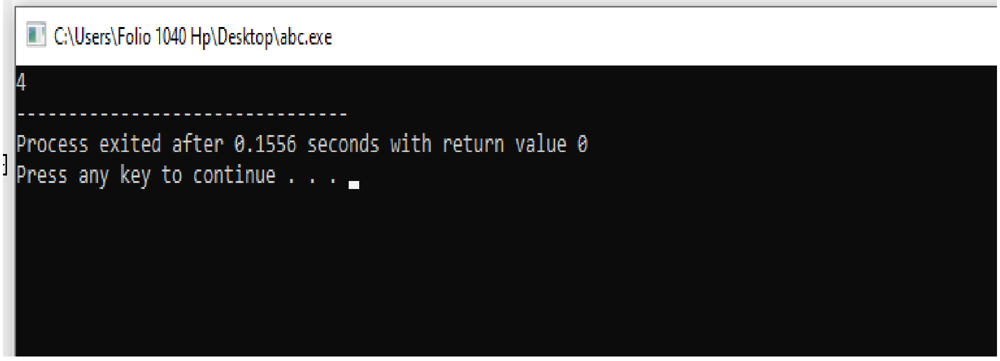


```
Ahmed
Fiza

--------------------------------
Process exited after 0.1478 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 6:

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

int main() {
  list<string> names = {"Taibah", "Ahmed", "fiza", "maira"};
  cout << names.size();
  return 0;
}
```

# OUTPUT:



```
4
--------------------------------
Process exited after 0.1556 seconds with return value 0
Press any key to continue . . .
```

# LAB NO 5

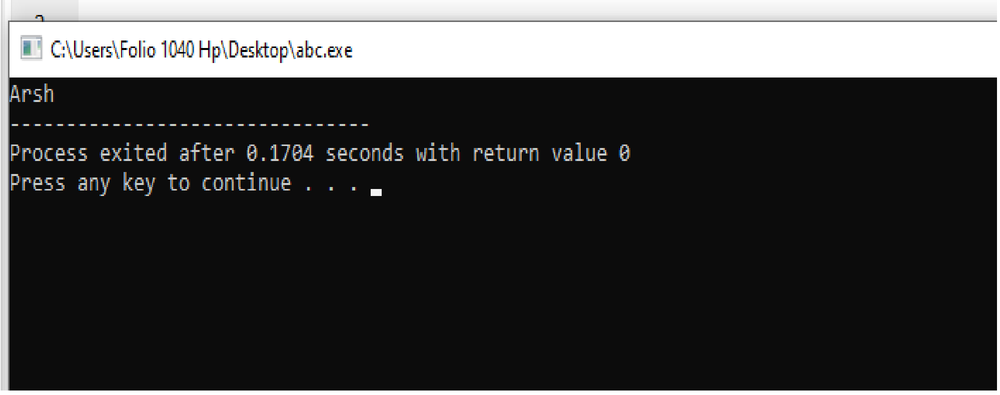
## STACK

A stack stores multiple elements in a specific order, called LIFO.
LIFO stands for Last in, First Out. To visualize LIFO, think of a pile of pancakes, where pancakes are both added and removed from the top. So when removing a pancake, it will always be the last one you added. This way of organizing elements is called LIFO in computer science and programming.

## PROGRAM NO 1:

```
#include <stack>
#include<iostream>
Using namespace std;
stack<string> names;
stack<string> names = {"Arsh", "Ahmed", "fiza", "maira"};
  cout << names.top();
  return 0;
}
```

**OUTPUT:**

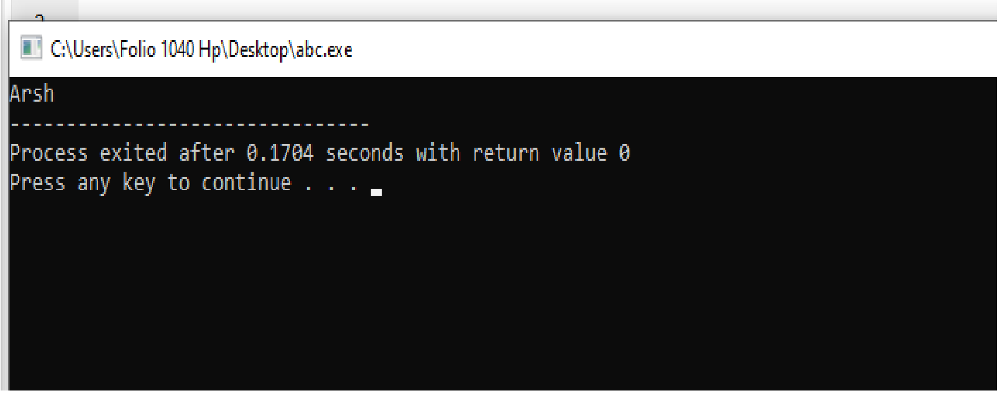


```
Arsh
--------------------------------
Process exited after 0.1704 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2:

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

int main() {
  // Create a stack of strings

  stack<string> names;

  // Add elements to the stack
  names.push("Taibah");
  names.push("Ahmed");
  names.push("fiza");
  names.push("maira");
  // Change the value of the top element
  names.top() = "arsh";

  // Access the top element
  cout << names.top();
  return 0;
}
```

## OUTPUT:

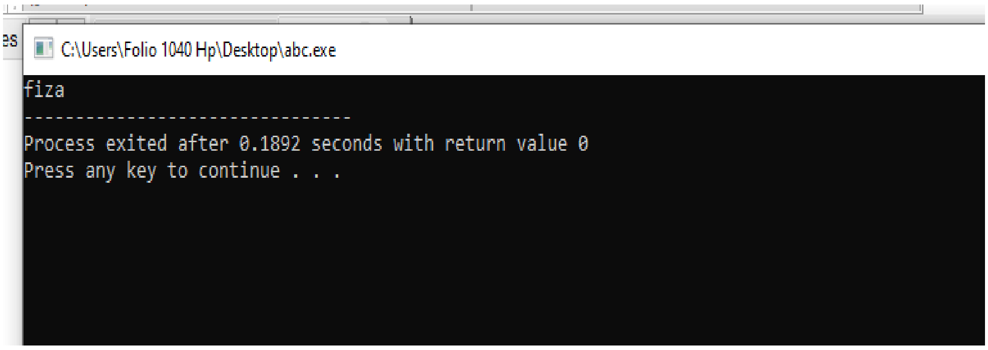


```
Arsh
--------------------------------
Process exited after 0.1704 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 3

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

int main() {
  // Create a stack of strings called cars
  stack<string> names;

  // Add elements to the stack
  names.push("Taibah");
  names.push("Ahmed");
  names.push("fiza");
  names.push("maira");

  // Remove the last/latest added element
  names.pop();

  // Access the top element
  cout << names.top();
  return 0;
}
```

## OUTPUT:

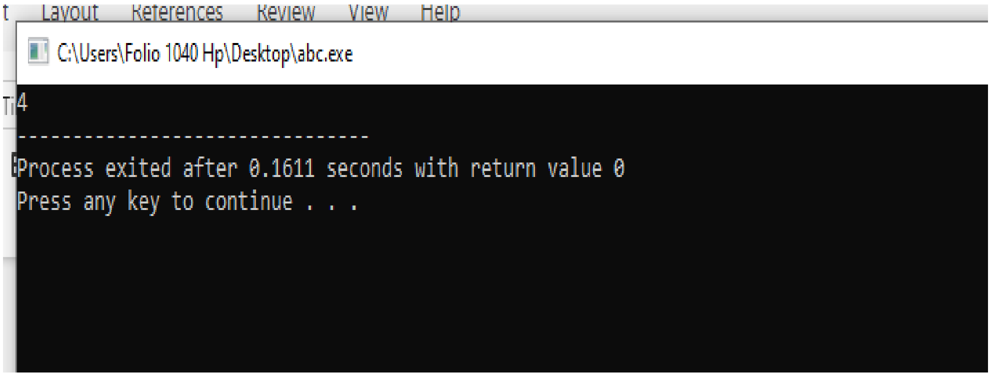


```
fiza
--------------------------------
Process exited after 0.1892 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 4:

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

int main() {
  // Create a stack of strings called cars
  stack<string> names;

  // Add elements to the stack
  names.push("Taibah");
  names.push("Ahmed");
  names.push("fiza");
  names.push("maira");
    // Get the size of the stack
  cout << names.size();
  return 0;
}
```

## OUTPUT:



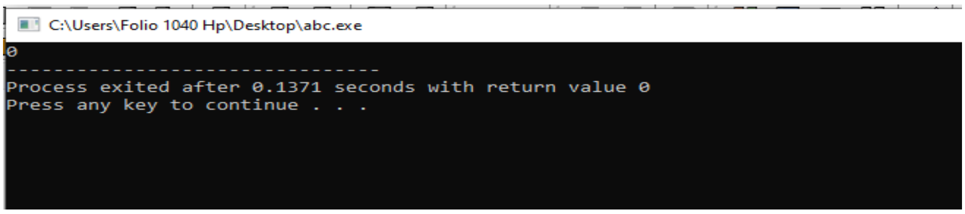
```
4
--------------------------------
Process exited after 0.1611 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 5:

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

int main() {
  // Create a stack of strings called cars
  stack<string> names;

  // Add elements to the stack
```

```
 names.push("Taibah");
 names.push("Ahmed");
 names.push("fiza");
 names.push("maira");
   // Get the size of the stack
 cout << names.empty();
 return 0;
}
```

# OUTPUT:



C:\Users\Folio 1040 Hp\Desktop\abc.exe

```
0
--------------------------------
Process exited after 0.1371 seconds with return value 0
Press any key to continue . . .
```

# LAB NO 6

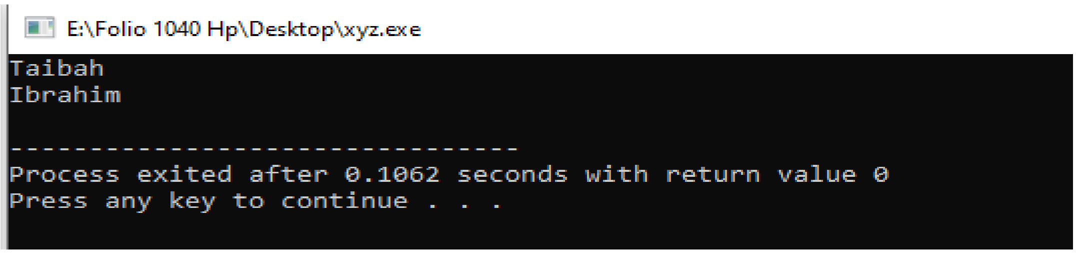
# QUEUE

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.

## PROGRAM NO 1:

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

int main() {
// Create a queue of strings
queue<string> names;

// Add elements to the queue
names.push("Taibah");
names.push("Fiza");
 names.push("Ahmed");
names.push("Ibrahim");

// Access the front element (first and oldest)
cout << names.front() << "\n";

// Access the back element (last and newest)
cout << names.back() << "\n";
return 0;
}
```

## OUTPUT:

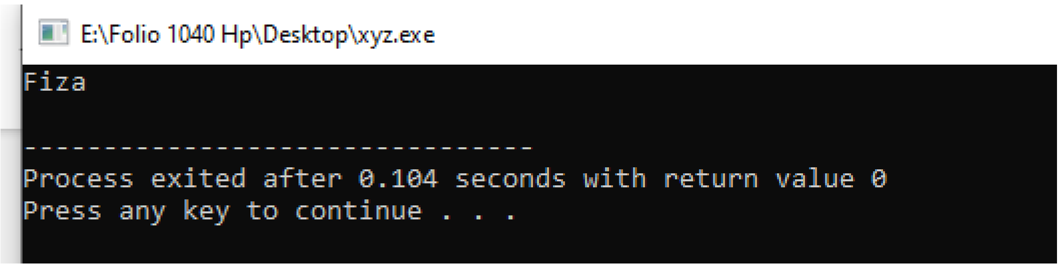


```
Taibah
Ibrahim

--------------------------------
Process exited after 0.1062 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2:

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

// Create a queue of strings
queue<string> names;
// Add elements to the queue
names.push("Taibah");
names.push("Fiza");
names.push("Ahmed");
names.push("Ibrahim");

// Remove the front element
names.pop();

// Access the front element (first and oldest)
cout << names.front() << "\n";
return 0;
}
```

**OUTPUT:**



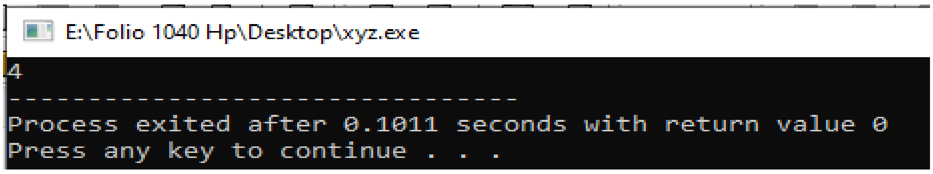
E:\Folio 1040 Hp\Desktop\xyz.exe

```
Fiza

--------------------------------
Process exited after 0.104 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
// Create a queue of strings
queue<string> names;
// Add elements to the queue
names.push("Taibah");
names.push("Fiza");
names.push("Ahmed");
names.push("Ibrahim");
// Get the size of the queue
cout << names.size();

return 0;
}
```

**OUTPUT**



E:\Folio 1040 Hp\Desktop\xyz.exe

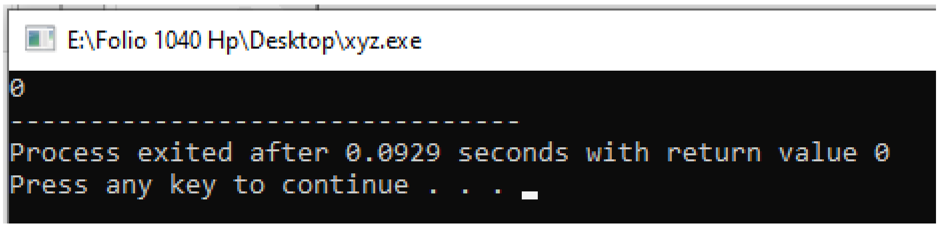
```
4
--------------------------------
Process exited after 0.1011 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4:

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

```
int main() {

// Create a queue of strings
queue<string> names;

// Add elements to the queue
names.push("Taibah");
names.push("Fiza");
names.push("Ahmed");
names.push("Ibrahim");

// Check if the queue is empty
cout << names.empty(); return 0;
}
```

**OUTPUT:**



E:\Folio 1040 Hp\Desktop\xyz.exe

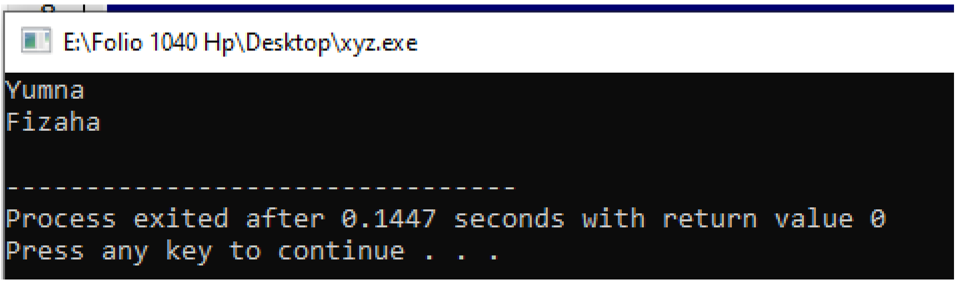
```
0
--------------------------------
Process exited after 0.0929 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 5

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

int main() {

// Create a queue of strings
queue<string> names;

// Add elements to the queue
names.push("Taibah");
names.push("Fiza");
names.push("Ahmed");
names.push("Ibrahim");

// Change the value of the front element
names.front() = "Yumna";

// Change the value of the back element
names.back() = "Fizaha";

// Access the front element (first and oldest)
cout << names.front() << "\n";

// Access the back element (last and newest)
cout << names.back() << "\n";

return 0;
```

}

**OUTPUT**



```
Yumna
Fizaha

--------------------------------
Process exited after 0.1447 seconds with return value 0
Press any key to continue . . .
```

# LAB NO 7

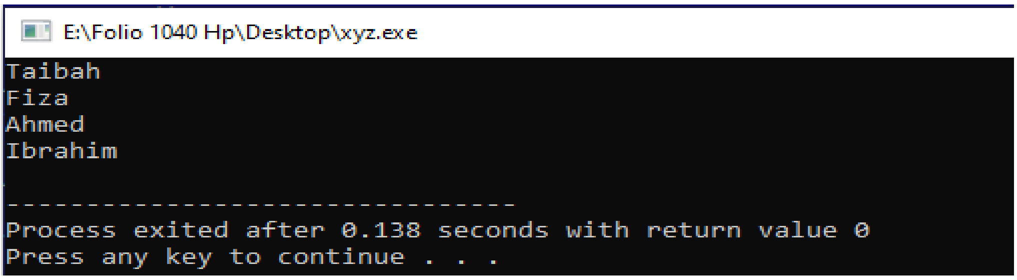
## DEQUE

A deque (stands for **d**ouble-**e**nded **queue**) however, is more flexible, as elements can be added and removed from both ends (at the front and the back). You can also access elements by index numbers.

## PROGRAM NO 1:

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

int main() {

// Create a deque called names that will store strings
deque<string> names = {"Taibah","Fiza", "Ahmed", "Ibrahim"};

// Print deque elements

for (string name : names) { cout << name << "\n";
}
return 0;
}
```

**OUTPUT**

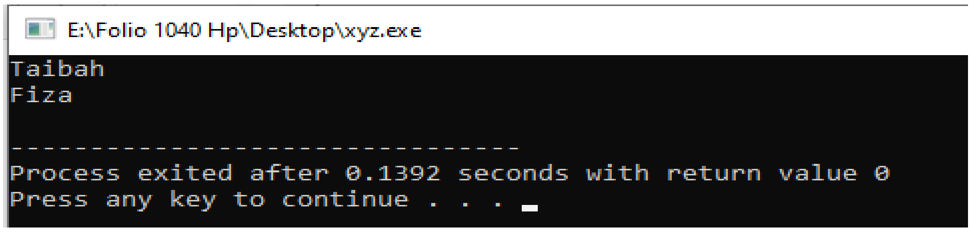


```
Taibah
Fiza
Ahmed
Ibrahim

--------------------------------
Process exited after 0.138 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2

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

int main() {

// Create a deque called names that will store strings
deque<string> names = {"Taibah","Fiza", "Ahmed", "Ibrahim"};

// Get the first element
cout << names[0] << "\n";
// Get the second element
cout << names[1] << "\n"
return 0;
}
```

**OUTPUT**

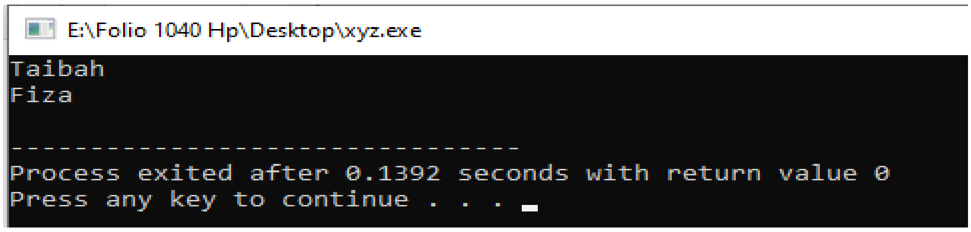


```
Taibah
Fiza

--------------------------------
Process exited after 0.1392 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3

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

int main() {

// Create a deque called names that will store strings
deque<string> names = {"Taibah","Fiza", "Ahmed", "Ibrahim"};

// Get the first element
cout << names.front<< "\n";
// Get the second element
cout << names.back << "\n";
return 0;
}
```

**OUTPUT**

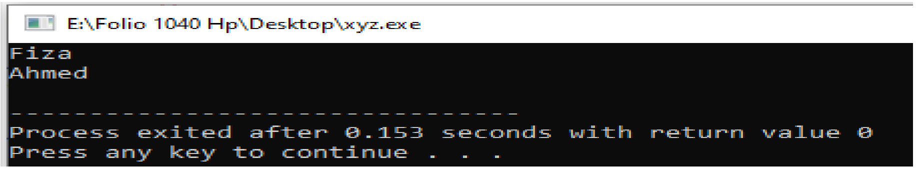


```
Taibah
Fiza

--------------------------------
Process exited after 0.1392 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4

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

int main() {

// Create a deque called names that will store strings
deque<string> names = {"Taibah","Fiza", "Ahmed", "Ibrahim"};

// Get the second element
cout << names.at(1) << "\n";

// Get the third element
cout << names.at(2) << "\n";"
return 0;
 }
```

## OUTPUT

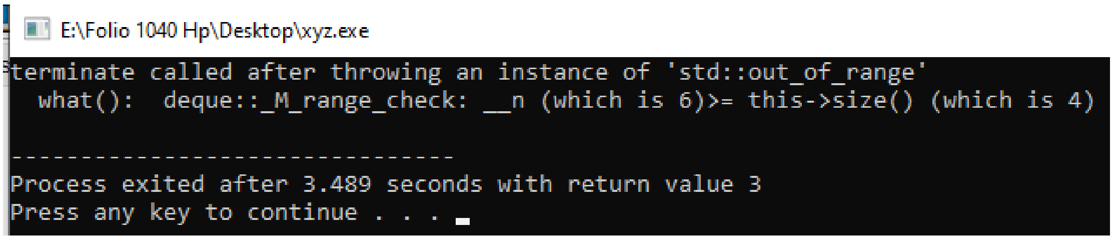


```
Fiza
Ahmed

--------------------------------
Process exited after 0.153 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 5

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

int main() {

// Create a deque called names that will store strings
deque<string> names = {"Taibah","Fiza", "Ahmed", "Ibrahim"};
// Try to access an element that does not exist (will throw an exception)
cout << cars.at(6)
return 0;
 }
```

## OUTPUT



```
terminate called after throwing an instance of 'std::out_of_range'
  what():  deque::_M_range_check: __n (which is 6)>= this->size() (which is 4)

--------------------------------
Process exited after 3.489 seconds with return value 3
Press any key to continue . . .
```

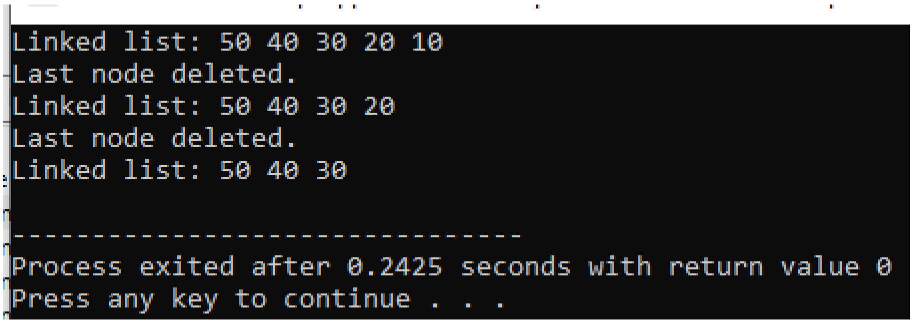
# LAB NO 8

# Linked list

## PROGRAM NO 1

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

// Node structure for the linked list
struct Node {
    int data;
    Node* next;
};

// Pointer to the head of the list
Node* head = nullptr;

// Function to insert a node at the start of the linked list
void insertAtStart(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

// Function to delete the last node of the linked list
void deleteAtEnd() {
    // Check if the list is empty
    if (head == nullptr) {
        cout << "List is empty. No nodes to delete." << endl;
        return;
    }

    // If there is only one node in the list
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        cout << "Last node deleted." << endl;
        return;
    }

    // Traverse to the second-to-last node
    Node* current = head;
    while (current->next != nullptr && current->next->next != nullptr) {
        current = current->next;
    }

    // Delete the last node
    Node* temp = current->next;
    current->next = nullptr;
    delete temp;

    cout << "Last node deleted." << endl;
}

// Function to display the linked list
void display() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    cout << "Linked list: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Insert elements at the start
    insertAtStart(10);
    insertAtStart(20);
    insertAtStart(30);
    insertAtStart(40);
    insertAtStart(50);

    // Display the current list
    display();

    // Delete the last node
    deleteAtEnd();

    // Display the list after deletion
    display();

    // Delete the last node again
    deleteAtEnd();

    // Display the list after second deletion
    display();

    return 0;
}
```

**OUTPUT**

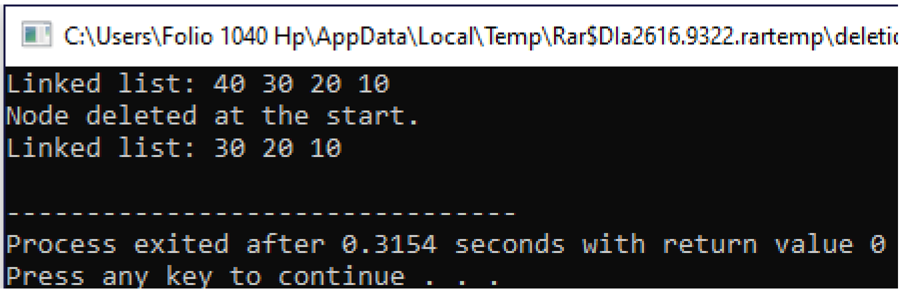


```
Linked list: 50 40 30 20 10
Last node deleted.
Linked list: 50 40 30 20
Last node deleted.
Linked list: 50 40 30

--------------------------------
Process exited after 0.2425 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2

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

// Node structure for the linked list
struct Node {
    int data;
    Node* next;
};

// Pointer to the head of the list
Node* head = nullptr;

// Function to insert a node at the start of the linked list
void insertAtStart(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

// Function to delete the node at the start of the linked list
void deleteAtStart() {
    // Check if the list is empty
    if (head == nullptr) {
        cout << "List is empty. No nodes to delete." << endl;
        return;
    }

    // Store the current head node
    Node* temp = head;
    // Move head to the next node
    head = head->next;

    // Delete the old head node
    delete temp;

    cout << "Node deleted at the start." << endl;
}

// Function to display the linked list
void display() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    cout << "Linked list: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Insert elements at the start
    insertAtStart(10);
    insertAtStart(20);
    insertAtStart(30);
    insertAtStart(40);

    // Display the current list
    display();

    // Delete the node at the start
    deleteAtStart();

    // Display the list after deletion
    display();

    return 0;
}
```

**OUTPUT**



C:\Users\Folio 1040 Hp\AppData\Local\Temp\Rar$DIa2616.9322.rartemp\deleti

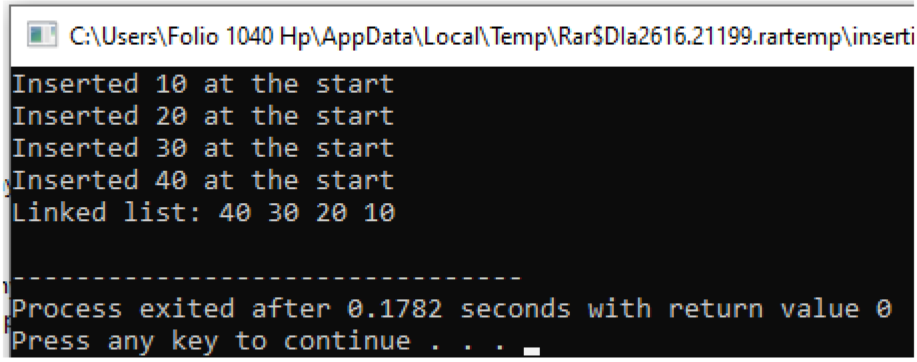
```
Linked list: 40 30 20 10
Node deleted at the start.
Linked list: 30 20 10

--------------------------------
Process exited after 0.3154 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3

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

// Node structure for the linked list
struct Node {
    int data;
    Node* next;
};
```

```
// Pointer to the head of the list
Node* head = nullptr;

// Function to insert a node at the start of the linked list
void insertAtStart(int value) {
    // Create a new node
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head; // Point the new node to the current head
    head = newNode;       // Update head to the new node
    cout << "Inserted " << value << " at the start" << endl;
}

// Function to display the linked list
void display() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    cout << "Linked list: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Insert elements at the start
    insertAtStart(10);
    insertAtStart(20);
    insertAtStart(30);
    insertAtStart(40);

    // Display the linked list
    display();

    return 0;
}
```

**OUTPUT**



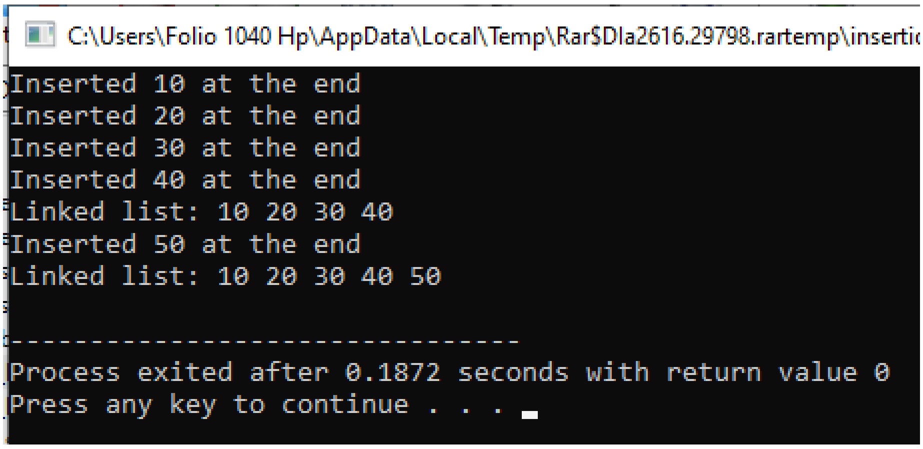
C:\Users\Folio 1040 Hp\AppData\Local\Temp\Rar$DIa2616.21199.rartemp\inserti

```
Inserted 10 at the start
Inserted 20 at the start
Inserted 30 at the start
Inserted 40 at the start
Linked list: 40 30 20 10

--------------------------------
Process exited after 0.1782 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4

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

// Node structure for the linked list
struct Node {
    int data;
    Node* next;
};

// Pointer to the head of the list
Node* head = nullptr;

// Function to insert a node at the end of the linked list
void insertAtEnd(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr; // New node will be the last node, so `next` is set to
nullptr

    // If the list is empty, the new node becomes the head
    if (head == nullptr) {
        head = newNode;
        cout << "Inserted " << value << " at the end" << endl;
        return;
    }

    // Traverse to the end of the list
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    // Link the last node to the new node
    temp->next = newNode;
    cout << "Inserted " << value << " at the end" << endl;
}

// Function to display the linked list
void display() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    cout << "Linked list: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Insert elements at the end
    insertAtEnd(10);
    insertAtEnd(20);
    insertAtEnd(30);
    insertAtEnd(40);

    // Display the current list
    display();

    // Insert another element at the end
    insertAtEnd(50);

    // Display the list after the new insertion
    display();

    return 0;
}
```

**OUTPUT**



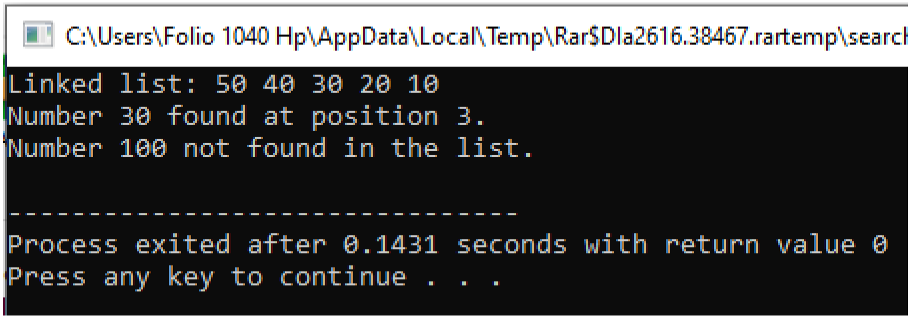
C:\Users\Folio 1040 Hp\AppData\Local\Temp\Rar$DIa2616.29798.rartemp\inserti

```
Inserted 10 at the end
Inserted 20 at the end
Inserted 30 at the end
Inserted 40 at the end
Linked list: 10 20 30 40
Inserted 50 at the end
Linked list: 10 20 30 40 50

--------------------------------
Process exited after 0.1872 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 5

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

// Node structure for the linked list
struct Node {
    int data;
    Node* next;
};

// Pointer to the head of the list
Node* head = nullptr;

// Function to insert a node at the start of the linked list
void insertAtStart(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

// Function to search for a number in the linked list
void search(int value) {
    // Traverse the list to find the value
    Node* current = head;
    int position = 1; // Position starts from 1 (for human-friendly indexing)

    while (current != nullptr) {
        if (current->data == value) {
            cout << "Number " << value << " found at position " << position << "." << endl;
            return; // Exit once the number is found
        }
        current = current->next;
        position++;
    }

    // If the value is not found in the list
    cout << "Number " << value << " not found in the list." << endl;
}

// Function to display the linked list
void display() {
    if (head == nullptr) {
        cout << "List is empty" << endl;
        return;
    }
    Node* temp = head;
    cout << "Linked list: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Insert elements at the start
    insertAtStart(10);
    insertAtStart(20);
    insertAtStart(30);
    insertAtStart(40);
    insertAtStart(50);

    // Display the current list
    display();

    // Search for a number in the linked list
    search(30); // Number found in the list
    search(100); // Number not found in the list

    return 0;
}
```

**OUTPUT**



C:\Users\Folio 1040 Hp\AppData\Local\Temp\Rar$DIa2616.38467.rartemp\searc

```
Linked list: 50 40 30 20 10
Number 30 found at position 3.
Number 100 not found in the list.

--------------------------------
Process exited after 0.1431 seconds with return value 0
Press any key to continue . . .
```

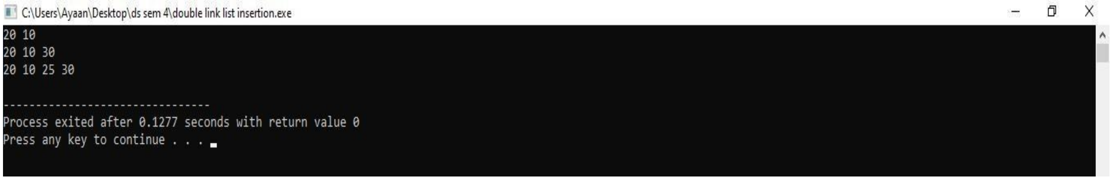
# LAB NO 9

# Doubly linked list

## PROGRAM NO 1

```
// insert node at the front
void insertFront(struct Node** head, int data) {

   // allocate memory for newNode
   struct Node* newNode = new Node;

   // assign data to newNode
   newNode->data = data;

   // point next of newNode to the first node of the doubly linked list
   newNode->next = (*head);

   // point prev to NULL
   newNode->prev = NULL;

   // point previous of the first node (now first node is the second node) to newNode
   if ((*head) != NULL)
      (*head)->prev = newNode;

   // head points to newNode
   (*head) = newNode;
}
```

**OUTPUT**



C:\Users\Ayaan\Desktop\ds sem 4\double link list insertion.exe

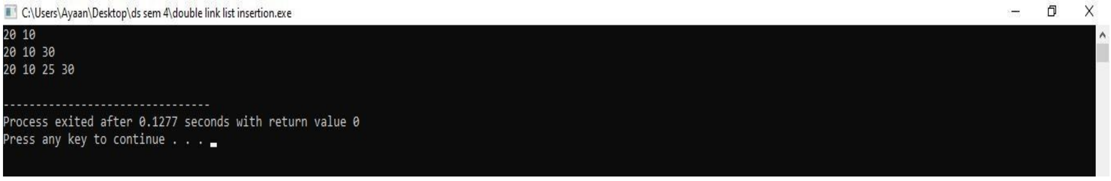
```
20 10
20 10 30
20 10 25 30

--------------------------------
Process exited after 0.1277 seconds with return value 0
Press any key to continue . . .
```

## PROGRAM NO 2

```
// insert a node after a specific node
void insertAfter(struct Node* prev_node, int data) {

   // check if previous node is NULL
   if (prev_node == NULL) {
      cout << "previous node cannot be NULL";
      return;
   }

   // allocate memory for newNode
   struct Node* newNode = new Node;

   // assign data to newNode
   newNode->data = data;
```

```
    // set next of newNode to next of prev node
    newNode->next = prev_node->next;

    // set next of prev node to newNode
    prev_node->next = newNode;

    // set prev of newNode to the previous node
    newNode->prev = prev_node;

    // set prev of newNode's next to newNode
    if (newNode->next != NULL)
        newNode->next->prev = newNode;
}
```

**OUTPUT**



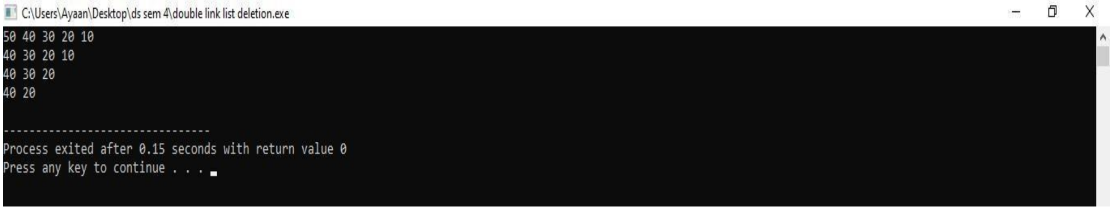
C:\Users\Ayaan\Desktop\ds sem 4\double link list insertion.exe

```
20 10
20 10 30
20 10 25 30

--------------------------------
Process exited after 0.1277 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3

```
// delete a node from the doubly linked list
void deleteNode(struct Node** head, struct Node* del_node) {
  // if head or del is null, deletion is not possible
  if (*head == NULL || del_node == NULL)
    return;

  // if del_node is the head node, point the head pointer to the next of del_node
  if (*head == del_node)
    *head = del_node->next;

  // if del_node is not at the last node, point the prev of node next to del_node to the previous of del_node
  if (del_node->next != NULL)
    del_node->next->prev = del_node->prev;

  // if del_node is not the first node, point the next of the previous node to the next node of del_node
  if (del_node->prev != NULL)
    del_node->prev->next = del_node->next;

  // free the memory of del_node
  free(del_node);
}
```

**OUTPUT**



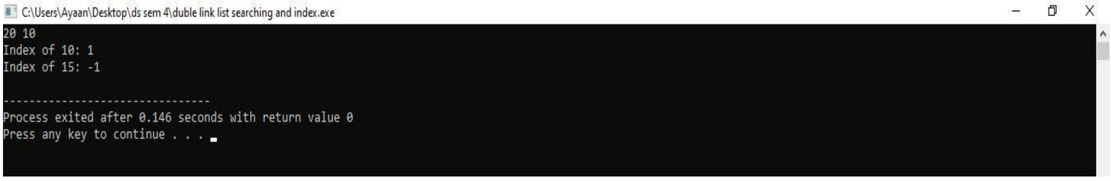
C:\Users\Ayaan\Desktop\ds sem 4\double link list deletion.exe

```
50 40 30 20 10
40 30 20 10
40 30 20
40 20

--------------------------------
Process exited after 0.15 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM N 4: SEARCHING AND FINDING INDEX:

```
// Search for an element
int search(int value) {
Node *temp = head;
int index = 0;
while (temp != nullptr) {
if (temp->data == value) { return index;
}
temp = temp->next; index++;
}
return -1;
}
// Find the index of an element int findIndex(int value) {
return search(value);
}

// Traverse and display the list void traverse() {
Node *temp = head; while (temp != nullptr) {
cout << temp->data << " "; temp = temp->next;
}
cout << endl;
}
int main() {
// Separate tests for each function

// Test Insertions insertFront(10); insertFront(20); traverse(); // 20 10
// Test Search and Edit
cout << "Index of 10: " << search(10) << endl; // 0
// Test Finding Index
cout << "Index of 15: " << findIndex(15) << endl; // 0


return 0;
}
```

**OUTPUT**



C:\Users\Ayaan\Desktop\ds sem 4\duble link list searching and index.exe

```
20 10
Index of 10: 1
Index of 15: -1

--------------------------------
Process exited after 0.146 seconds with return value 0
Press any key to continue . . .
```

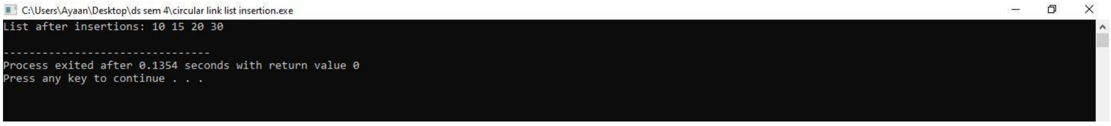
# <u>LAB NO 10</u>

# circular linked list

## PROGRAM NO 1

```
#include <iostream>

using namespace std;

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

struct Node* addToEmpty(struct Node* last, int data) {
  if (last != NULL)
return last;

  // allocate memory to the new node
  struct Node* newNode = (struct Node*)
malloc(sizeof(struct Node));

  // assign data to the new node
newNode->data = data;

  // assign last to newNode
  last = newNode;

  // create link to iteself
  last->next = last;

  return last;
}

// add node to the front
struct Node* addFront(struct Node* last, int data) {
  // check if the list is empty
  if (last == NULL) return addToEmpty(last, data);

  // allocate memory to the new node
  struct Node* newNode = (struct Node*)
malloc(sizeof(struct Node));

  // add data to the node
  newNode->data = data;

  // store the address of the current first node in the newNode
  newNode->next = last->next;

  // make newNode as head
  last->next = newNode;

  return last;
}
```

**OUTPUT**



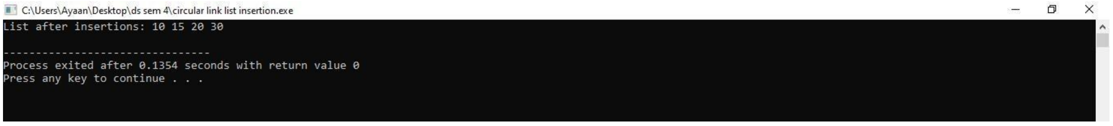
C:\Users\Ayaan\Desktop\ds sem 4\circular link list insertion.exe

```
List after insertions: 10 15 20 30

--------------------------------
Process exited after 0.1354 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 2

```
// insert node after a specific node
struct Node* addAfter(struct Node* last, int data, int item) {
  // check if the list is empty
  if (last == NULL) return NULL;
  struct Node *newNode, *p;

  p = last->next;
  do {
  // if the item is found, place newNode after it
  if (p->data == item) {
    // allocate memory to the new node
   newNode = (struct Node*)malloc(sizeof(struct Node));

    // add data to the node
    newNode->data = data;

    // make the next of the current node as the next of newNode
    newNode->next = p->next;

    // put newNode to the next of p
    p->next = newNode;

    // if p is the last node, make newNode as the last node
    if (p == last) last = newNode;
    return last;
  }

  p = p->next;
  } while (p != last->next);

  cout << "\nThe given node is not present in the list" << endl;
  return last;
}
```

**OUTPUT**



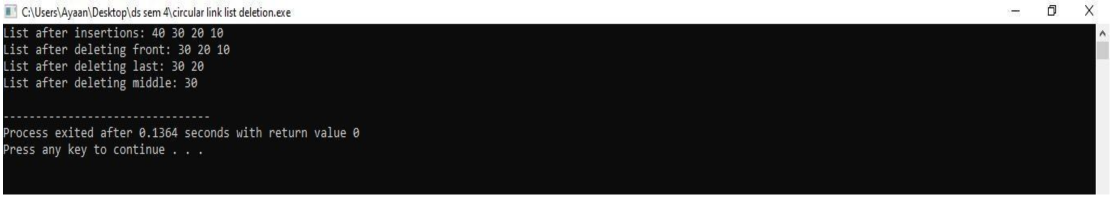
C:\Users\Ayaan\Desktop\ds sem 4\circular link list insertion.exe

```
List after insertions: 10 15 20 30

--------------------------------
Process exited after 0.1354 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3

```
// delete a node
void deleteNode(Node** last, int key) {
  // if linked list is empty
  if (*last == NULL) return;

  // if the list contains only a single node
  if ((*last)->data == key && (*last)->next == *last) {
  free(*last);
  *last = NULL;
  return;
  }

  Node *temp = *last, *d;

  // if last is to be deleted
  if ((*last)->data == key) {
  // find the node before the last node
  while (temp->next != *last) temp = temp->next;
  // point temp node to the next of last i.e. first node
  temp->next = (*last)->next;
  free(*last);
  *last = temp->next;
  }

  // travel to the node to be deleted
  while (temp->next != *last && temp->next->data != key) {
  temp = temp->next;
  }

  // if node to be deleted was found
  if (temp->next->data == key) {
  d = temp->next;
  temp->next = d->next;
  free(d);
  }
}
```

**OUTPUT**



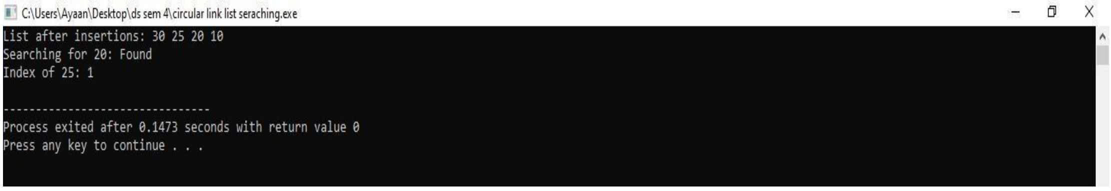
C:\Users\Ayaan\Desktop\ds sem 4\circular link list deletion.exe

```
List after insertions: 40 30 20 10
List after deleting front: 30 20 10
List after deleting last: 30 20
List after deleting middle: 30

--------------------------------
Process exited after 0.1364 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 4

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

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

Node* head = NULL;

// Insert at the Front
void insertFront(int value) {
Node* newNode = new Node();
newNode->data = value;
if (head == NULL) {
newNode->next = newNode;
head = newNode;
} else {
Node* temp = head;
while (temp->next != head) {
temp = temp->next;
}
newNode->next = head;
temp->next = newNode;
head = newNode;
}
}
// Search for an Element
bool search(int value) {
if (head == NULL) return false;
Node* temp = head;
do {
if (temp->data == value) return true;
temp = temp->next;
} while (temp != head);
return false;
}
// Find the Index of an Element
int findIndex(int value) {
if (head == NULL) return -1;
Node* temp = head;
int index = 0; do {
if (temp->data == value)
return index;
temp = temp->next;
index++;
} while (temp != head);
}
}
int main() { insertFront(10); insertFront(20); insertFront(25); insertFront(30);
cout << "List after insertions: "; traverse();
cout << "Searching for 20: " << (search(20) ? "Found" : "Not Found") << endl; cout
<< "Index of 25: " << findIndex(25) << endl;

return 0;
}
```

**OUTPUT**


C:\Users\Ayaan\Desktop\ds sem 4\circular link list seraching.exe

```
List after insertions: 30 25 20 10
Searching for 20: Found
Index of 25: 1

--------------------------------
Process exited after 0.1473 seconds with return value 0
Press any key to continue . . .
```

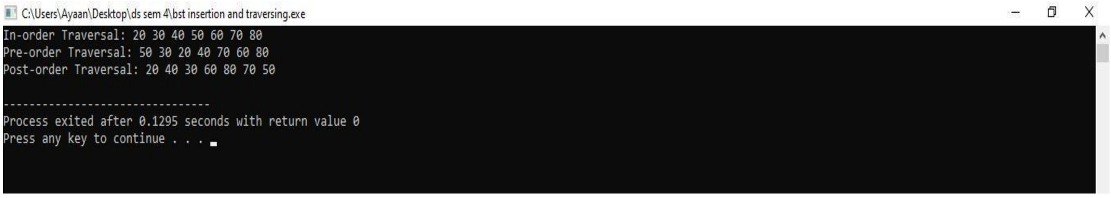
# LAB NO 11

# BINARY SEARCH TREE

## PROGRAM NO 1: INSERTION AND TRAVERSING (IN,PRE AND POST ORDER)

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

struct Node { int data; Node* left; Node* right;
};

// Function to create a new node
Node* createNode(int value) {
Node* newNode = new Node;
newNode->data = value;
 newNode->left = nullptr;
newNode->right = nullptr;
return newNode;
}

// Function to insert a value into the BST
Node* insert(Node* root, int value) {
if (root == nullptr) {
return createNode(value);
}
if (value < root->data) {
root->left = insert(root->left, value);
} else if (value > root->data) {
root->right = insert(root->right, value);
}
return root;
}
// In-order traversal (Left, Root, Right)
void inOrder(Node* root) {
if (root != nullptr) {
inOrder(root->left);
cout << root->data << " ";
inOrder(root->right);
}
}

// Pre-order traversal (Root, Left, Right)
void preOrder(Node* root) {
if (root != nullptr) {
cout << root->data << " ";
 preOrder(root->left);
 preOrder(root->right);
}
}

// Post-order traversal (Left, Right, Root)
void postOrder(Node* root) {
if (root != nullptr) {
 postOrder(root->left);
postOrder(root->right);
cout << root->data << " ";
}
}
int main() {
Node* root = nullptr;

// Insert nodes
root = insert(root, 50);
root = insert(root, 30);
root = insert(root, 70);
root = insert(root, 20);
 root = insert(root, 40);
root = insert(root, 60);
root = insert(root, 80);

cout << "In-order Traversal: ";
inOrder(root);
cout << endl;

cout << "Pre-order Traversal: ";
PreOrder(root);
cout << endl;

cout << "Post-order Traversal: ";
postOrder(root);
cout << endl;
return 0;
}
```

**OUTPUT**


C:\Users\Ayaan\Desktop\ds sem 4\bst insertion and traversing.exe

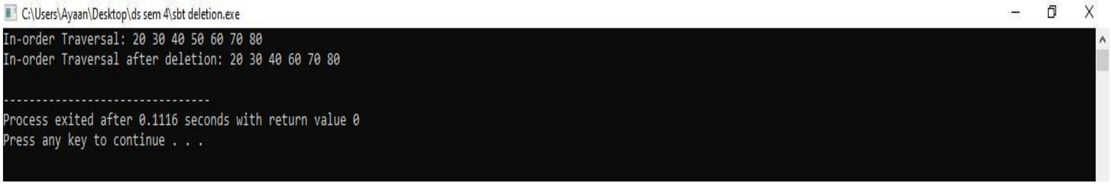
```
In-order Traversal: 20 30 40 50 60 70 80
Pre-order Traversal: 50 30 20 40 70 60 80
Post-order Traversal: 20 40 30 60 80 70 50

--------------------------------
Process exited after 0.1295 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 2

```
// Deleting a node
struct node *deleteNode(struct node *root, int key) {
  // Return if the tree is empty
  if (root == NULL) return root;

  // Find the node to be deleted
  if (key < root->key)
```

```
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);
  else {
    // If the node is with only one child or no child
    if (root->left == NULL) {
      struct node *temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      struct node *temp = root->left;
      free(root);
      return temp;
    }

    // If the node has two children
    struct node *temp = minValueNode(root->right);

    // Place the inorder successor in position of the node to be deleted
    root->key = temp->key;

    // Delete the inorder successor
    root->right = deleteNode(root->right, temp->key);
  }
  return root;
}
// Driver code
int main() {
struct node *root = NULL;
  root = insert(root, 8);
  root = insert(root, 3);
  root = insert(root, 1);
  root = insert(root, 6);
  root = insert(root, 7);
  root = insert(root, 10);
  root = insert(root, 14);
  root = insert(root, 4);

  cout << "Inorder traversal: ";
  inorder(root);

  cout << "\nAfter deleting 10\n";
  root = deleteNode(root, 10);
  cout << "Inorder traversal: ";
  inorder(root);
}
```

**OUTPUT**

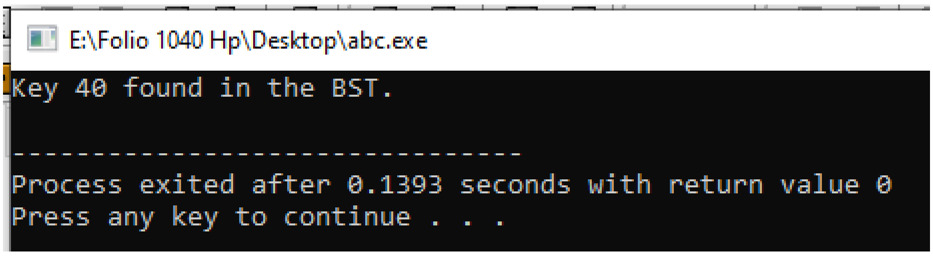


```
In-order Traversal: 20 30 40 50 60 70 80
In-order Traversal after deletion: 20 30 40 60 70 80

--------------------------------
Process exited after 0.1116 seconds with return value 0
Press any key to continue . . .
```

# PROGRAM NO 3

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

// Define the structure of a node in the BST
struct Node {
    int data;
    Node* left;
    Node* right;

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

// Function to insert a value into the BST
Node* insert(Node* root, int val) {
    if (root == nullptr) {
        return new Node(val);
    }
    if (val < root->data) {
        root->left = insert(root->left, val);
    } else if (val > root->data) {
        root->right = insert(root->right, val);
    }
    return root;
}

// Function to search for a value in the BST
bool search(Node* root, int key) {
    if (root == nullptr) {
        return false; // Key not found
    }
    if (root->data == key) {
        return true; // Key found
    }
    if (key < root->data) {
        return search(root->left, key);
    } else {
        return search(root->right, key);
    }
}

int main() {
    Node* root = nullptr;

    // Insert values into the BST
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 60);
    root = insert(root, 80);
    // Search for a value in the BST
    int key = 40;
    if (search(root, key)) {
        cout << "Key " << key << " found in the BST." << endl;
    } else {
        cout << "Key " << key << " not found in the BST." << endl;
    }

    return 0;
}
```

**OUTPUT**



```
Key 40 found in the BST.

--------------------------------
Process exited after 0.1393 seconds with return value 0
Press any key to continue . . .
```