DATA STRUCTURES(MC-201)

LABORATORY MANUAL

DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT OF APPLIED MATHEMATICS

SUBMITTED TO:-

Ms. Himanshi Lohit

Mr. Kriss Gunjan

SUBMITTED BY:-

Gurpreet Singh

23/MC/058

EXPERIMENTS LIST AND INDEX

Exp No.	OBJECTIVE	DATE	SIGNATURE
1	(a) linear search and binary search in an array (b) insertion and deletion in an array		
2	(a) Fetch a substring from a string, find its position and length of the substring. (b) Replacement of 1 string with another (c) Concatenation of 2 strings		
3	(a) Implement stack using arrays (push & pop) (b) Evaluate arithmetic expression by converting it from infix to postfix (c) Check for balanced parenthesis in an expression		
4	(a) Implement circular queue (b) Implement priority queue		

EXPERIMENT-1

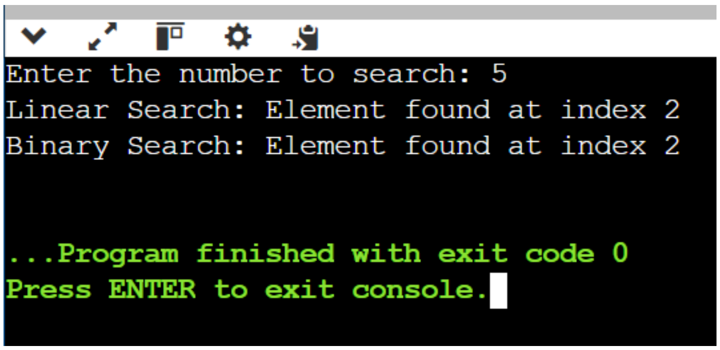
AIM- Linear Search and Binary Search in an array.

CODE-

```
#include <iostream>

using namespace std;


// Function for Linear Search

int linearSearch(int arr[], int size, int key) {

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

      if (arr[i] == key)

         return i; // Return index if key is found

   }

   return -1; // Return -1 if key is not found

}


// Function for Binary Search

int binarySearch(int arr[], int size, int key) {

   int low = 0, high = size - 1;

   while (low <= high) {

      int mid = (low + high) / 2;

      if (arr[mid] == key)

         return mid; // Return index if key is found

      else if (arr[mid] < key)

         low = mid + 1;

      else

         high = mid - 1;
```

```
    }
    return -1; // Return -1 if key is not found
}

int main() {
    int arr[] = {1, 3, 5, 7, 9, 11, 13, 15};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key;

    cout << "Enter the number to search: ";
    cin >> key;

    // Linear Search
    int result = linearSearch(arr, size, key);
    if (result != -1)
        cout << "Linear Search: Element found at index " << result << endl;
    else
        cout << "Linear Search: Element not found" << endl;

    // Binary Search
    result = binarySearch(arr, size, key);
    if (result != -1)
        cout << "Binary Search: Element found at index " << result << endl;
    else
        cout << "Binary Search: Element not found" << endl;

    return 0;
}
```

OUTPUT –



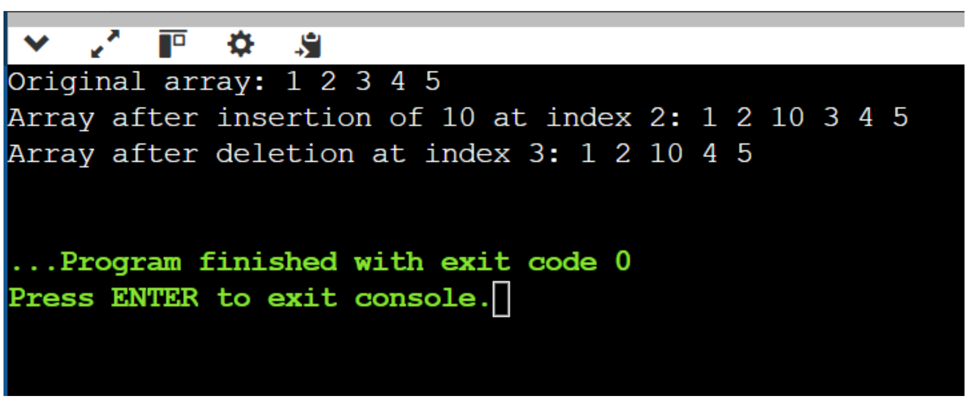
(b) **AIM**: Insertion and Deletion in an Array.

CODE-

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

// Function to insert an element at a specific position in an array
void insertElement(int arr[], int& size, int element, int position) {
   // Shift elements to the right to make space for the new element
   for (int i = size; i > position; i--) {
      arr[i] = arr[i - 1];
   }
   arr[position] = element; // Insert the element
   size++; // Increase the size of the array
}

// Function to delete an element from a specific position in an array
void deleteElement(int arr[], int& size, int position) {
   // Shift elements to the left to remove the element
   for (int i = position; i < size - 1; i++) {
      arr[i] = arr[i + 1];
   }
   size--; // Decrease the size of the array
}

// Function to print the array
void printArray(int arr[], int size) {
   for (int i = 0; i < size; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
}
```

```
int main() {
    int arr[100] = {1, 2, 3, 4, 5}; // Initial array
    int size = 5; // Initial size of the array

    cout << "Original array: ";
    printArray(arr, size);

    // Insertion example
    int element = 10;
    int position = 2; // Insert at index 2
    insertElement(arr, size, element, position);
    cout << "Array after insertion of " << element << " at index " << position << ": ";
    printArray(arr, size);

    // Deletion example
    position = 3; // Delete element at index 3
    deleteElement(arr, size, position);
    cout << "Array after deletion at index " << position << ": ";
    printArray(arr, size);

    return 0;
}
```

OUTPUT-



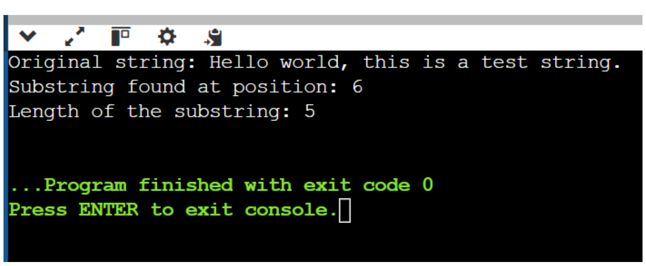
EXPERIMENT-2

(a) **AIM-**Fetch a substring from a string , find its position and length of the substring.

CODE-

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

// Function to fetch a substring and find its position and length
void fetchSubstring(const string& str, const string& subStr) {
  size_t pos = str.find(subStr); // Find position of substring
  if (pos != string::npos) {
    cout << "Substring found at position: " << pos << endl;
    cout << "Length of the substring: " << subStr.length() << endl;
  } else {
    cout << "Substring not found." << endl;
  }
}

int main() {
  string str = "Hello world, this is a test string.";
  string subStr = "world";

  // Fetching a substring
  cout << "Original string: " << str << endl;
  fetchSubstring(str, subStr);

  return 0;
}
```

OUTPUT-

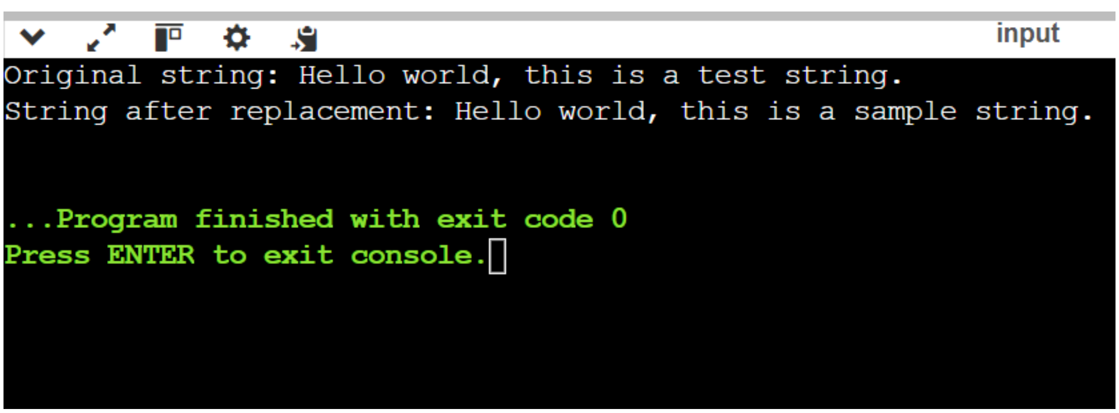


(b) Replacement of 1 string with another

CODE-

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

// Function to replace a substring with another
string replaceSubstring(string str, const string& toReplace, const string& replaceWith) {
  size_t pos = str.find(toReplace);
  while (pos != string::npos) {
    str.replace(pos, toReplace.length(), replaceWith); // Replace substring
    pos = str.find(toReplace, pos + replaceWith.length()); // Continue searching
  }
  return str;
}

int main() {
  string str = "Hello world, this is a test string.";
  cout << "Original string: " << str << endl;

  // Replacing a substring
  string replacedStr = replaceSubstring(str, "test", "sample");
  cout << "String after replacement: " << replacedStr << endl;

  return 0;
}
```

OUTPUT-

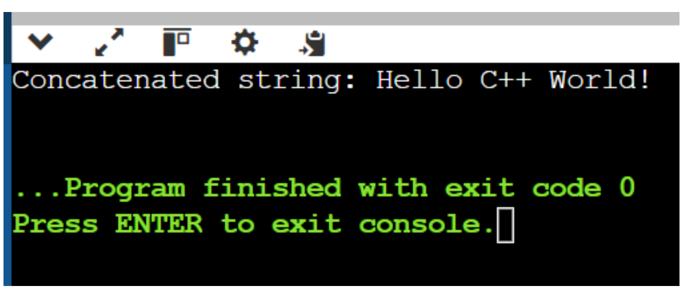


(c) **AIM-** Concatenation of two strings.

CODE-

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

// Function to concatenate two strings
string concatenateStrings(const string& str1, const string& str2) {
    return str1 + str2; // Concatenate and return
}

int main() {
    string str1 = "Hello ";
    string str2 = "C++ World!";

    // Concatenating two strings
    string concatenatedStr = concatenateStrings(str1, str2);
    cout << "Concatenated string: " << concatenatedStr << endl;

    return 0;
}
```

OUTPUT-



EXPERIMENT -3

(a) **AIM:** Implement stack using arrays(push and pop)

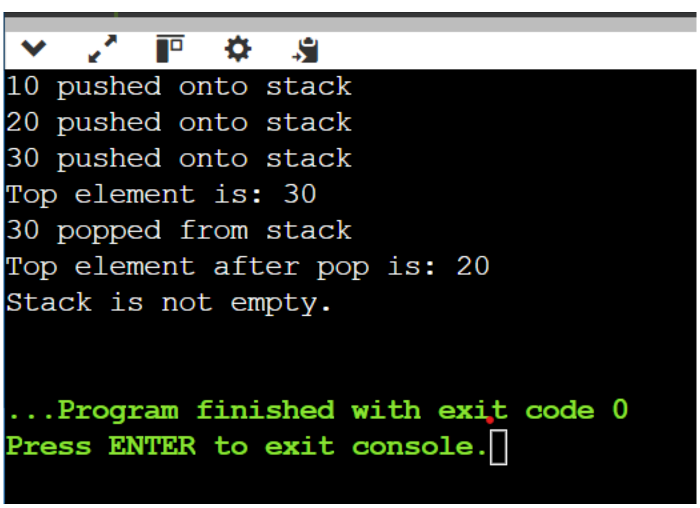
CODE:-

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

#define MAX 100 // Maximum size of the stack

class Stack {
private:
   int arr[MAX]; // Array to store stack elements
   int top;      // Points to the top element of the stack

public:
   Stack() { top = -1; } // Constructor initializes stack to be empty

   // Function to push an element onto the stack
   void push(int x) {
      if (top >= MAX - 1) {
         cout << "Stack Overflow!" << endl;
      } else {
         arr[++top] = x;
         cout << x << " pushed onto stack" << endl;
      }
```

```
    }

    // Function to pop the top element from the stack
    void pop() {
        if (top < 0) {
            cout << "Stack Underflow!" << endl;
        } else {
            int popped = arr[top--];
            cout << popped << " popped from stack" << endl;
        }
    }

    // Function to check the top element of the stack
    int peek() {
        if (top < 0) {
            cout << "Stack is Empty!" << endl;
            return -1;
        } else {
            return arr[top];
        }
    }

    // Function to check if the stack is empty
    bool isEmpty() {
        return (top < 0);
    }
};

int main() {
```

```
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);

    cout << "Top element is: " << s.peek() << endl;

    s.pop();
    cout << "Top element after pop is: " << s.peek() << endl;

    if (s.isEmpty()) {
        cout << "Stack is empty." << endl;
    } else {
        cout << "Stack is not empty." << endl;
    }

    return 0;
}
```

OUTPUT-



```
10 pushed onto stack
20 pushed onto stack
30 pushed onto stack
Top element is: 30
30 popped from stack
Top element after pop is: 20
Stack is not empty.


...Program finished with exit code 0
Press ENTER to exit console.
```

(b) **AIM:** Evaluate arithmetic expression by converting it from infix to postfix.

CODE:-

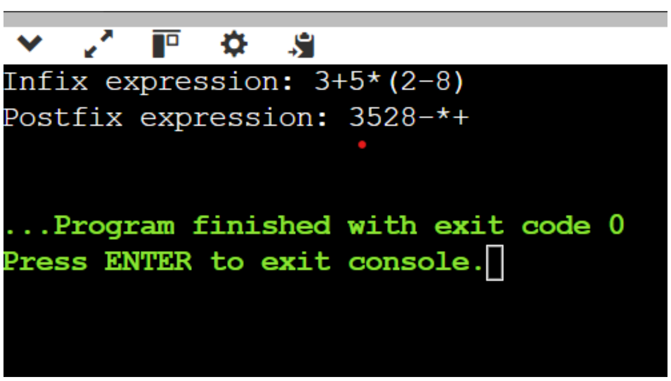
```
#include <iostream>

#include <stack>

#include <cctype> // For isdigit()

using namespace std;


// Function to return precedence of operators

int precedence(char op) {

    if (op == '+' || op == '-') {

        return 1;

    } else if (op == '*' || op == '/') {

        return 2;

    }

    return 0;

}


// Function to check if the character is an operator

bool isOperator(char c) {

    return (c == '+' || c == '-' || c == '*' || c == '/');
```

```
}

// Function to convert infix expression to postfix
string infixToPostfix(string infix) {
    stack<char> s; // Stack to hold operators
    string postfix = "";

    for (int i = 0; i < infix.length(); i++) {
        char c = infix[i];

        // If the character is an operand, add it to
postfix string
        if (isdigit(c)) {
            postfix += c;
        }
        // If the character is '(', push it to the stack
        else if (c == '(') {
            s.push(c);
        }
        // If the character is ')', pop and output from
the stack until '(' is encountered
        else if (c == ')') {
            while (!s.empty() && s.top() != '(') {
```

```
            postfix += s.top();
            s.pop();
        }
        s.pop(); // Pop '('
    }
    // If the character is an operator
    else if (isOperator(c)) {
        while (!s.empty() && precedence(s.top()) >=
precedence(c)) {
            postfix += s.top();
            s.pop();
        }
        s.push(c);
    }
}

// Pop all remaining operators from the stack
while (!s.empty()) {
    postfix += s.top();
    s.pop();
}

return postfix;
```

```
}


int main() {
    string infix = "3+5*(2-8)";
    cout << "Infix expression: " << infix << endl;


    string postfix = infixToPostfix(infix);
    cout << "Postfix expression: " << postfix << endl;


    return 0;
}
```

OUTPUT-



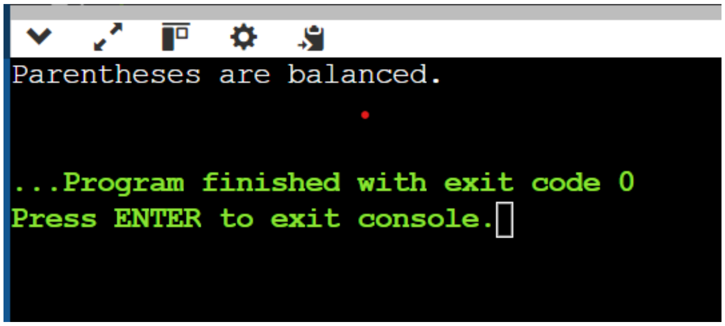
(c) **AIM:** Check for balanced parenthesis in an expression.

CODE:-

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

// Function to check if the parentheses are balanced
bool areParenthesesBalanced(string expr) {
    stack<char> s;

    // Traverse the given expression
    for (int i = 0; i < expr.length(); i++) {
        char c = expr[i];

        // If an opening bracket is found, push it to the stack
        if (c == '(' || c == '{' || c == '[') {
            s.push(c);
        }
        // If a closing bracket is found, check for matching opening brackets
        else if (c == ')' || c == '}' || c == ']') {
            // If the stack is empty, parentheses are not balanced
            if (s.empty()) {
                return false;
            }

            char top = s.top();
            s.pop();

            // Check if the popped bracket matches with the closing one
            if ((c == ')' && top != '(') || (c == '}' && top != '{') || (c == ']' && top != '[')) {
                return false;
            }
        }
    }

    // If the stack is empty, parentheses are balanced; otherwise, they are not
    return s.empty();
}

int main() {
    string expr = "{[()]}"; // Example expression

    if (areParenthesesBalanced(expr)) {
```

```
        cout << "Parentheses are balanced." << endl;
    } else {
        cout << "Parentheses are not balanced." << endl;
    }

    return 0;
}
```

OUTPUT-



EXPERIMENT-4

(a) **AIM:** Implement Circular Queue.

CODE:-

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

#define SIZE 5 // Define the maximum size of the queue

class CircularQueue {
private:
    int items[SIZE]; // Array to store the queue elements
    int front, rear; // Pointers to track the front and rear of the queue

public:
    // Constructor to initialize the queue
    CircularQueue() {
        front = -1;
```

```
    rear = -1;
}

// Function to check if the queue is full
bool isFull() {
    return (front == 0 && rear == SIZE - 1) || (front == rear + 1);
}

// Function to check if the queue is empty
bool isEmpty() {
    return front == -1;
}

// Function to add an element to the queue (enqueue)
void enqueue(int element) {
    if (isFull()) {
        cout << "Queue is full!" << endl;
        return;
    }

    if (front == -1) {
        front = 0; // If the queue was empty, set front to 0
    }

    rear = (rear + 1) % SIZE; // Update rear in a circular manner
    items[rear] = element;
    cout << element << " enqueued to the queue." << endl;
}

// Function to remove an element from the queue (dequeue)
int dequeue() {
    if (isEmpty()) {
        cout << "Queue is empty!" << endl;
        return -1;
    }

    int element = items[front]; // Get the front element

    // If the queue has only one element, reset it
    if (front == rear) {
        front = rear = -1;
    } else {
        front = (front + 1) % SIZE; // Update front in a circular manner
    }

    cout << element << " dequeued from the queue." << endl;
    return element;
}
```

```
    // Function to display the elements of the queue
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return;
        }

        cout << "Queue elements are: ";
        int i = front;
        while (i != rear) {
            cout << items[i] << " ";
            i = (i + 1) % SIZE;
        }
        cout << items[rear] << endl;
    }
};

int main() {
    CircularQueue q;

    // Enqueue elements
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);

    // Display elements
    q.display();

    // Dequeue elements
    q.dequeue();
    q.dequeue();

    // Display elements after dequeue
    q.display();

    // Enqueue more elements to test circular property
    q.enqueue(60);
    q.enqueue(70);

    // Display elements again
    q.display();

    return 0;
}
```

OUTPUT-

```
10 enqueued to the queue.
20 enqueued to the queue.
30 enqueued to the queue.
40 enqueued to the queue.
50 enqueued to the queue.
Queue elements are: 10 20 30 40 50
10 dequeued from the queue.
20 dequeued from the queue.
Queue elements are: 30 40 50
60 enqueued to the queue.
70 enqueued to the queue.
Queue elements are: 30 40 50 60 70
```

(b) **AIM:** Implement Priority Queue.

CODE:-

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

class PriorityQueue {
private:
   vector<int> heap; // Vector to store heap elements

   // Function to maintain the heap property (heapify up)
   void heapifyUp(int index) {
      while (index > 0) {
         int parent = (index - 1) / 2;
         if (heap[index] > heap[parent]) {
            swap(heap[index], heap[parent]); // Swap with parent
```

```
        index = parent; // Move to parent index
      } else {
        break; // If no swap needed, exit
      }
    }
  }

  // Function to maintain the heap property (heapify down)
  void heapifyDown(int index) {
    int size = heap.size();
    while (index < size) {
      int largest = index; // Assume current index is largest
      int leftChild = 2 * index + 1; // Left child index
      int rightChild = 2 * index + 2; // Right child index

      // Check if left child exists and is greater than the largest
      if (leftChild < size && heap[leftChild] > heap[largest]) {
        largest = leftChild;
      }

      // Check if right child exists and is greater than the largest
      if (rightChild < size && heap[rightChild] > heap[largest]) {
        largest = rightChild;
      }

      // If the largest is not the current index, swap and continue
      if (largest != index) {
        swap(heap[index], heap[largest]);
        index = largest; // Move to the largest index
```

```
            } else {
                break; // If already in correct position, exit
            }
        }
    }

public:
    // Function to insert an element into the priority queue
    void enqueue(int value) {
        heap.push_back(value); // Add element to the end
        heapifyUp(heap.size() - 1); // Restore the heap property
        cout << value << " enqueued to priority queue" << endl;
    }

    // Function to remove and return the highest priority element
    int dequeue() {
        if (heap.empty()) {
            cout << "Priority Queue is empty!" << endl;
            return -1; // Indicate an invalid operation
        }
        int maxElement = heap[0]; // The highest priority element
        heap[0] = heap.back(); // Move the last element to the root
        heap.pop_back(); // Remove the last element
        heapifyDown(0); // Restore the heap property
        return maxElement; // Return the highest priority element
    }

    // Function to display the elements of the priority queue
    void display() {
```

```
        cout << "Priority Queue elements: ";
        for (int value : heap) {
            cout << value << " ";
        }
        cout << endl;
    }
};

int main() {
    PriorityQueue pq;

    pq.enqueue(10);
    pq.enqueue(20);
    pq.enqueue(15);
    pq.enqueue(30);
    pq.enqueue(40);

    pq.display(); // Display current elements in the queue

    cout << "Dequeued: " << pq.dequeue() << endl; // Remove and display highest priority element
    pq.display(); // Display remaining elements
     cout << "Dequeued: " << pq.dequeue();}
```

OUTPUT-

```
10 enqueued to priority queue
20 enqueued to priority queue
15 enqueued to priority queue
30 enqueued to priority queue
40 enqueued to priority queue
Priority Queue elements: 40 30 15 10 20
Dequeued: 40
Priority Queue elements: 30 20 15 10
Dequeued: 30

...Program finished with exit code 0
Press ENTER to exit console.
```