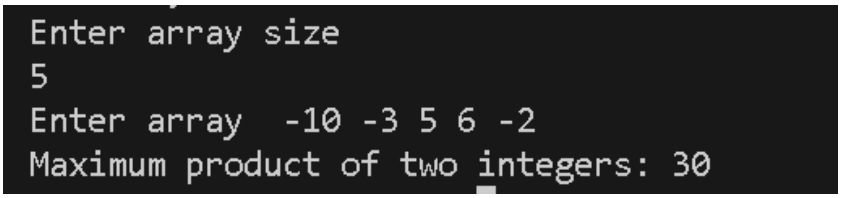
Department of Computer Science and Engineering

CSE2122 (Data Structure Lab)

1.Given an integer array, find the maximum product of two integers in it. For example, consider array A with (-10, -3, 5, 6, -2}. The maximum product is the (-10, -3) or (5, 6) pair.

```
#include <bits/stdc++.h>
using namespace std;

int maxProduct(int arr[], int size)
{
   if (size < 2)
   {
      cout << "Enter two element" << endl;
      return 0;
   }

//INT_MIN initialized with the smallest possible int value

   int max_product = INT_MIN;
   int second_max_product = INT_MIN;

   for (int i = 0; i < size; i++)
   {
      if (arr[i] > max_product)
      {
         second_max_product = max_product;
         max_product = arr[i];
      }
      else if (arr[i] > second_max_product)
      {
         second_max_product = arr[i];
      }
   }

   return max_product * second_max_product;
}
```

```
int main()
{
   // take array size as n and arr[n] size.
   int n;
   cin >> n;
   int arr[n];
   for (int i = 0; i < n; i++)
   {
      cin >> arr[i];
   }
   // function call and result store in result variable
   int result = maxProduct(arr, n);
   cout << "Maximum product of two integers: " << result << endl;
}
```



2. Insert a node to its correct sorted position in a sorted linked list. First, create a linked list with some sorted data, then ask for any one data to be inserted to its correct position.

```
#include <iostream>

using namespace std;

struct Node
{
   int data;
   Node *next;
   Node(int value) : data(value), next(nullptr) {}
};

void insertILinkedList(Node *&head, int value)
{
   Node *newNode = new Node(value);

   if (head == nullptr || value < head->data)
   {
      // If the list is empty or the new value is smaller than the head, insert at the beginning
      newNode->next = head;
      head = newNode;
      return;
   }

   Node *current = head;
```

```
    while (current->next != nullptr && current->next->data < value)
    {
        current = current->next;
    }

    // Insert the new node after the current node
    newNode->next = current->next;
    current->next = newNode;
}

void printLinkedList(Node *head)
{
    Node *current = head;
    while (current != nullptr)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main()
{
    Node *head = nullptr;

    insertILinkedList(head, 2);
    insertILinkedList(head, 4);
    insertILinkedList(head, 6);
    insertILinkedList(head, 8);
    insertILinkedList(head, 10);

    cout << "Sorted Linked List: ";
    printLinkedList(head);

    int valueToInsert;
    cout << "Enter a value to insert into the sorted linked list: ";
    cin >> valueToInsert;

    insertILinkedList(head, valueToInsert);

    cout << "Updated Sorted Linked List: ";
    printLinkedList(head);

    return 0;  }
```

```
Sorted Linked List: 2 4 6 8 10
Enter a value to insert into the sorted linked list: 5
Updated Sorted Linked List: 2 4 5 6 8 10
```

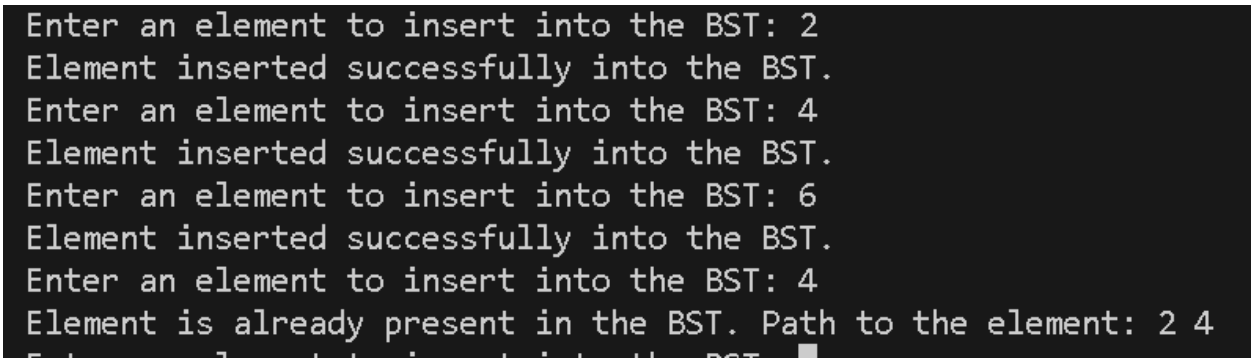
3. Write a program to insert an element in a Binary search tree; if the element already inserted before then display the location.

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

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

TreeNode *insert(TreeNode *root, int value, bool &alreadyInserted)
{
   if (root == nullptr)
   {
      return new TreeNode(value);
   }

   if (value < root->data)
   {
      // Insert into the left subtree
      root->left = insert(root->left, value, alreadyInserted);
   }
   else if (value > root->data)
   {
      // Insert into the right subtree
      root->right = insert(root->right, value, alreadyInserted);
   }
   else
   {
      // Element is already present
      alreadyInserted = true;
   }

   return root;
}
```

```
// Function to display the path to an element in the BST
void displayPathToElement(TreeNode *root, int value)
{
   if (root == nullptr)
   {
      return;
   }

   cout << root->data << " ";

   if (value < root->data)
   {
      displayPathToElement(root->left, value);
   }
   else if (value > root->data)
   {
      displayPathToElement(root->right, value);
   }
}
```

```
int main()
{
   TreeNode *root = nullptr;
   bool alreadyInserted = false;

   while (true)
   {
      int elementToInsert;
      cout << "Enter an element to insert into the BST: ";
      cin >> elementToInsert;

      root = insert(root, elementToInsert, alreadyInserted);

      if (alreadyInserted)
      {
         cout << "Element is already present in the BST. Path to the element: ";
         displayPathToElement(root, elementToInsert);
         cout << endl;
      }
```

```
    else
    {
        cout << "Element inserted successfully into the BST." << endl;
    }
  }

  return 0;
}
```



SET B

1. Split nodes of a linked list into the front and back halves. If the total number of elements in the list is odd, the extra element should go in the front list. For example, list (2, 3, 5, 7, 11) should yield the two lists with (2, 3, 5) and (7, 11).

```
#include <iostream>

using namespace std;

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

ListNode* createNode(int data) {
   ListNode* newNode = new ListNode;
   newNode->data = data;
   newNode->next = nullptr;
   return newNode;
}

void splitLinkedList(ListNode* head, ListNode*& front, ListNode*& back) {
   if (!head) {
      front = back = nullptr;
      return;
   }

   ListNode* slow = head;
   ListNode* fast = head;

   while (fast->next != nullptr && fast->next->next != nullptr) {
      slow = slow->next;
      fast = fast->next->next;
   }

   front = head;
   back = slow->next;
   slow->next = nullptr;
}
```

```
void printList(ListNode* head) {
   while (head != nullptr) {
      cout << head->data << " ";
      head = head->next;
   }
   cout << endl;
}

int main() {
   ListNode* head = createNode(2);
   head->next = createNode(3);
   head->next->next = createNode(5);
   head->next->next->next = createNode(7);
   head->next->next->next->next = createNode(11);

   ListNode* frontHalf;
   ListNode* backHalf;

   splitLinkedList(head, frontHalf, backHalf);

   printList(frontHalf);
   printList(backHalf);

   return 0;
}
```

2. Write a program to evaluate an arithmetic expression written in postfix notation. The postfix notation consists of single letter, single digit and arithmetic operators (+, -,,/, ^). If the expression contains a letter, you have to ask the value (single digit) for that letter.

```
#include <iostream>
#include <cstring>

using namespace std;

int evaluatePostfix(const char *postfixExpression)
{
   int stack[1000];
   int top = -1;
```

```
for (int i = 0; postfixExpression[i]; i++)
{
   char c = postfixExpression[i];

   if (isalpha(c))
   {
      int value;
      cout << "Enter the value of " << c << ": ";
      cin >> value;
      stack[++top] = value;
   }
   else if (isdigit(c))
   {
      stack[++top] = c - '0';
   }
   else if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
   {
      int operand2 = stack[top--];
      int operand1 = stack[top--];

      int result;
      switch (c)
      {
      case '+':
         result = operand1 + operand2;
         break;
      case '-':
         result = operand1 - operand2;
         break;
      case '*':
         result = operand1 * operand2;
         break;
      case '/':
         result = operand1 / operand2;
         break;
      case '^':
         result = 1;
         for (int j = 0; j < operand2; j++)
         {
            result *= operand1;
         }
         break;
      }
```

```
            stack[++top] = result;
        }
    }

    return stack[top];
}

int main()
{
    char postfixExpression[1000];
    cout << "Enter the postfix expression: ";
    cin >> postfixExpression;

    int result = evaluatePostfix(postfixExpression);
    cout << "Result: " << result << endl;

    return 0;
}
```

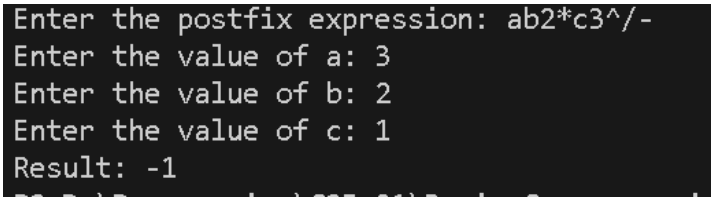
Input

Enter the value of a: 3

Enter the value of b: 2

Enter the value of c: 1

Output -1



3. Find out the shortest path of a Weighted Graph G with m nodes V1, V2, ...,Vm and weight of each edge is w(e) using Warshall's Algorithm.

```
#include <iostream>
#include <vector>
using namespace std;
#define INF 99999

void warshall(int graph[][100], int n)
{
   int dist[100][100];

   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         dist[i][j] = graph[i][j];
      }
   }

   for (int k = 0; k < n; k++)
   {
      for (int i = 0; i < n; i++)
      {
         for (int j = 0; j < n; j++)
         {
            if (dist[i][k] + dist[k][j] < dist[i][j])
            {
               dist[i][j] = dist[i][k] + dist[k][j];
            }
         }
      }
   }

   cout << "Shortest path distances between all pairs of vertices:" << endl;
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         if (dist[i][j] == INF)
         {
            cout << "INF ";
         }
```

```
            else
            {
                cout << dist[i][j] << " ";
            }
        }
        cout << endl;
    }
}

int main()
{
    int n, m;
    cout << "Enter the number of vertices (n): ";
    cin >> n;
    cout << "Enter the number of edges (m): ";
    cin >> m;

    int graph[100][100];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j)
            {
                graph[i][j] = 0;
            }
            else
            {
                graph[i][j] = INF;
            }
        }
    }

    cout << "Enter the edges and weights (V1 V2 weight):" << endl;
    for (int i = 0; i < m; i++)
    {
        int V1, V2, weight;
        cin >> V1 >> V2 >> weight;
        graph[V1][V2] = weight;
    }
 warshall(graph, n);

    return 0;     }
```