Aim: Stack Implementation.

Objectives: The main aim is to understand and implement the stack data structure and its operations such as push and pop using arrays. Additionally, apply the stack for practical problems like infix-to-postfix conversion, parenthesis balancing, and postfix expression evaluation.

Tools Used: Visual Studio Code.

Concept:

A stack 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.

Operations of Stack**:**

1) **Push (Insertion)**
   Push operation adds an element to the top of the stack.
   Example:
   Initial Stack: [10, 20, 30]
   Push 40:
   Resulting Stack: [10, 20, 30, 40]

Algorithm:

- Check if the stack is full. If full, return "Stack Overflow".
- Increment the top pointer.
- Insert the new element at the position pointed to by the top.

Steps:

1.) if top == MAX - 1: print("Stack Overflow")
2.) else: stack[++top] = element

2) **Pop (Deletion)**
   Pop operation removes and returns the topmost element of the stack.
   Example:
   Initial Stack: [10, 20, 30, 40]
   Pop:
   Resulting Stack: [10, 20, 30]

Algorithm:

- Check if the stack is empty. If empty, return "Stack Underflow".
- Access the topmost element.
- Decrement the top pointer.

Steps:

1.) if top == -1: print("Stack Underflow")
2.) else: element = stack[top--]

Problem Statement:

1. Implement Stack Using array.
2. Infix to Postfix using stack.
3. Balancing of parenthesis using Stack.
4. Postfix Evaluation.

Solution:

1.) Implement Stack Using array.

```
#include <iostream>

#define max 100

using namespace std;


int i = 0, n, ch, arr[max], key;


class Stack

{

public:

  void input()

  {

    cout << "Enter the size of the Stack: ";

    cin >> n;

  }


  void menu()

  {

    do

    {

      cout << endl
```

```
            << "Select options: ";
        cout << endl
            << "1 Push ";
        cout << endl
            << "2 Pop";
        cout << endl
            << "3 Display";
        cout << endl
            << "4 Exit" << endl
            << endl;
        cin >> ch;
        switch (ch)
        {
        case 1:
           Push();
           break;
        case 2:
           Pop();
           break;
        case 3:
           Display();
           break;
        case 4:
           break;
        default:
           cout << "Enter proper Option." << endl;
           break;
        }
     } while (ch != 4);
  }

  void Push()
```

```
{
    if (i >= n)
    {
        cout << "Stack Overflow!!!" << endl;
        return;
    }
    cout << endl
         << "Enter the Element you want to Push into the Stack" << endl;
    cin >> key;
    arr[i] = key;
    i++;
}

void Pop()
{
    if (i <= 0)
    {
        cout << "Stack Underflow!!!" << endl;
        return;
    }
    i--;
    cout << endl
         << "Popped element: " << arr[i] << endl;
}

void Display()
{
    if (i == 0)
    {
        cout << "Stack is empty." << endl;
        return;
    }
```

```
        cout << endl
             << "Displaying Stack" << endl;
        for (int j = i - 1; j >= 0; j--)
        {
            cout << arr[j] << endl;
        }
    }
};

int main()
{
    Stack sk;
    sk.input();
    sk.menu();
    return 0;
}
```

Output:

```
Enter the size of the Stack: 3

Select options:
1 Push
2 Pop
3 Display
4 Exit

1

Enter the Element you want to Push into the Stack
10

Select options:
1 Push
2 Pop
3 Display
4 Exit

1

Enter the Element you want to Push into the Stack
20

Select options:
1 Push
2 Pop
3 Display
4 Exit
```

```
1

Enter the Element you want to Push into the Stack
30

Select options:
1 Push
2 Pop
3 Display
4 Exit

1
Stack Overflow!!!

Select options:
1 Push
2 Pop
3 Display
4 Exit

2

Popped element: 30

Select options:
1 Push
2 Pop
3 Display
4 Exit
```

```
3

Displaying Stack
20
10

Select options:
1 Push
2 Pop
3 Display
4 Exit

2

Popped element: 20

Select options:
1 Push
2 Pop
3 Display
4 Exit

2

Popped element: 10

Select options:
1 Push
2 Pop
3 Display
```

```
Select options:
1 Push
2 Pop
3 Display
4 Exit

2

Popped element: 10

Select options:
1 Push
2 Pop
3 Display
4 Exit

2
Stack Underflow!!!

Select options:
1 Push
2 Pop
3 Display
4 Exit

4
```

2.) Infix to Postfix using stack.

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

string infixExpression;
int step = 1;

class InfixToPostfix
{
public:
   void input()
   {
      cout << "Enter an infix expression: ";
      cin >> infixExpression;
```

```
}

void conversion()
{
    string postfixExpression = infixToPostfix(infixExpression);
    cout << "\nPostfix expression: " << postfixExpression << endl;
}

int precedence(char op)
{
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    if (op == '^')
        return 3;
    return 0;
}

bool isOperator(char ch)
{
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

string infixToPostfix(string Q)
{
    stack<char> st;
    string P;

    // Step 1: Push '(' onto the stack and add ')' to the end of Q
    st.push('(');
```

```
    Q += ')';

    // Print table header
    cout << setw(10) << "Symbol"
         << setw(25) << "Stack"
         << setw(20) << "Expression" << endl;
    cout << string(70, '-') << endl;

    // Step 1: iterate Q from left to right
    for (char ch : Q)
    {
        // Step 3: If operand, add it to P
        if (isalnum(ch))
        {
            P += ch;
        }
        // Step 4: If left parenthesis, push onto stack
        else if (ch == '(')
        {
            st.push(ch);
        }
        // Step 5: If operator is encountered
        else if (isOperator(ch))
        {
            while (!st.empty() && precedence(st.top()) >= precedence(ch))
            {
                P += st.top();
                st.pop();
            }
            st.push(ch);
        }
```

```
        // Step 6: If right parenthesis is encountered
        else if (ch == ')')
        {
            while (!st.empty() && st.top() != '(')
            {
                P += st.top();
                st.pop();
            }
            st.pop(); // Remove the left parenthesis
        }

        // Print current step details
        cout << setw(10) << step++
             << setw(25) << getStackContents(st)
             << setw(20) << P << endl;
    }

    return P; // Return the postfix expression
  }

  string getStackContents(stack<char> st)
  {
    string contents;
    while (!st.empty())
    {
      contents += st.top();
      st.pop();
    }
    return contents;
  }
};
```

```
int main()
{
   InfixToPostfix itp;
   itp.input();
   itp.conversion();

   return 0;
}
```

Output:

```
Enter an infix expression: (a+b)*c+(d/e)
    Symbol                    Stack          Expression
-------------------------------------------------------------------
         1                      ((
         2                      ((                   a
         3                     +((                   a
         4                     +((                  ab
         5                       (                 ab+
         6                      *(                 ab+
         7                      *(                ab+c
         8                      +(               ab+c*
         9                     (+(               ab+c*
        10                     (+(              ab+c*d
        11                    /(+(              ab+c*d
        12                    /(+(             ab+c*de
        13                      +(            ab+c*de/
        14                                   ab+c*de/+

Postfix expression: ab+c*de/+
```

3.) Balancing of parenthesis using Stack.

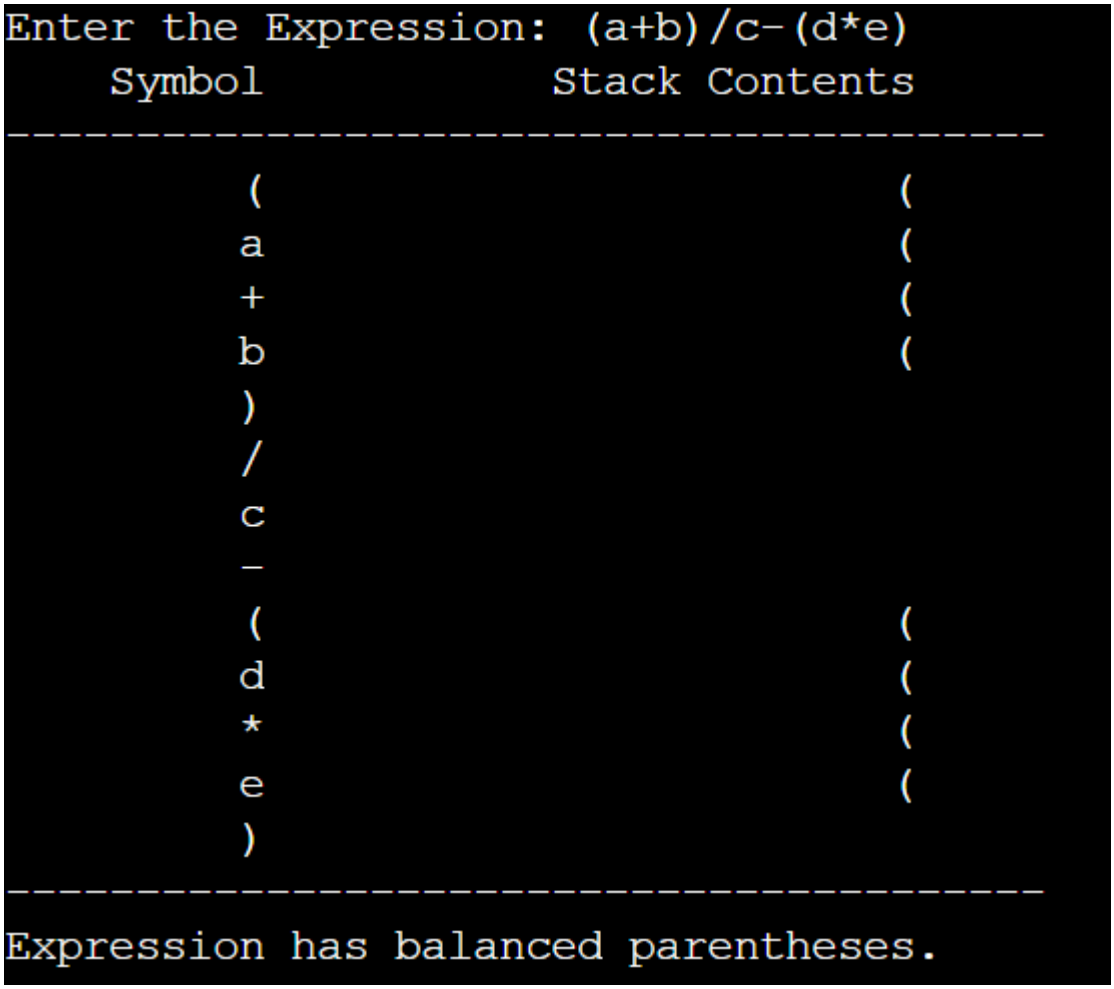
```
#include <iostream>
#include <stack>
#include <iomanip>
using namespace std;

class BalancingParenthesis
{
```

```
private:
    string exp;
    int step = 1;

    string getStackContents(stack<char> st)
    {
        string contents;
        while (!st.empty())
        {
            contents = st.top() + contents;
            st.pop();
        }
        return contents;
    }

public:
    void input()
    {
        cout << "Enter the Expression: ";
        cin >> exp;
    }

    void calculation()
    {
        stack<char> st;

        cout << setw(10) << "Symbol"
             << setw(25) << "Stack Contents"
             << endl;
        cout << string(40, '-') << endl;
```

```
    for (char ch : exp)
    {
        if (ch == '(')
        {
            st.push(ch);
        }
        else if (ch == ')')
        {
            if (!st.empty() && st.top() == '(')
            {
                st.pop();
            }
            else
            {
                st.push(ch);
            }
        }

        cout << setw(10) << ch
             << setw(25) << getStackContents(st)
             << endl;
    }

    cout << string(40, '-') << endl;
    if (st.empty())
    {
        cout << "Expression has balanced parentheses.\n";
    }
    else
    {
        cout << "Expression does not have balanced parentheses.\n";
```

```
    }
  }
};

int main()
{
  BalancingParenthesis bp;
  bp.input();
  bp.calculation();
  return 0;
}
```

Output:



4.) Postfix Evaluation.

```
#include <iostream>
#include <stack>
#include <iomanip>
#include <string>
#include <cctype>
using namespace std;

class PostfixEvaluation
{
private:
   string exp;

public:
   void input()
   {
      cout << "Enter the Postfix Expression: ";
      cin >> exp;
   }

   void conversion()
   {
      stack<int> st;

      cout << setw(10) << "Symbol"
          << setw(25) << "Stack Contents"
          << endl;
      cout << string(40, '-') << endl;

      for (char ch : exp)
      {
```

```
        if (isdigit(ch))
        {
            st.push(ch - '0');
        }
        else
        {
            if (st.size() < 2)
            {
                cout << "Error: Invalid Postfix Expression\n";
                return;
            }
            int b = st.top();
            st.pop();
            int a = st.top();
            st.pop();

            switch (ch)
            {
            case '+':
                st.push(a + b);
                break;
            case '-':
                st.push(a - b);
                break;
            case '*':
                st.push(a * b);
                break;
            case '/':
                if (b == 0)
                {
                    cout << "Error: Division by zero\n";
```

```
                return;
            }
            st.push(a / b);
            break;
        default:
            cout << "Error: Unknown operator " << ch << "\n";
            return;
        }
    }

    cout << setw(10) << ch
        << setw(25) << getStackContents(st)
        << endl;
  }

  cout << string(40, '-') << endl;
  if (st.size() == 1)
  {
    cout << "Result of the Postfix Expression: " << st.top() << endl;
  }
  else
  {
    cout << "Error: Invalid Postfix Expression\n";
  }
 }

private:
 string getStackContents(stack<int> st)
 {
  string contents;
  while (!st.empty())
```

```
        {
            contents = to_string(st.top()) + " " + contents;
            st.pop();
        }
        return contents;
    }
};

int main()
{
    PostfixEvaluation pe;
    pe.input();
    pe.conversion();
    return 0;
}
```

Output:

```
Enter the Postfix Expression: 23+5*
    Symbol            Stack Contents
----------------------------------------
         2                         2
         3                       2 3
         +                         5
         5                       5 5
         *                        25
----------------------------------------
Result of the Postfix Expression: 25
```

Observation: A stack's LIFO (Last In, First Out) nature makes it useful for many tasks. Using arrays for stack implementation is simple but limited by size. Converting infix to postfix uses the stack to handle operator precedence. Balancing parentheses is easily managed by pushing and popping brackets. For postfix evaluation, the stack stores numbers and calculates results step by step.