**EXPERIMENT NO. 03**

|  |  |
| --- | --- |
| **DATE OF PERFORMANCE:** | **GRADE:** |
| **DATE OF ASSESSMENT:** | **SIGNATURE OF LECTURER/ TTA:** |

**AIM: Implementation of Stack.**

**THEORY:**

**STACK:**

**A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. Stacks are sometimes known as LIFO (last in, first out) lists. As the items can be added or removed only from the top i.e. the last item to be added to a stack is the first item to be removed.**

**The two basic operations associated with stacks are:**

**• *Push*: is the term used to insert an element into a stack.**

**• *Pop*: is the term used to delete an element from a stack.**

**“Push” is the term used to insert an element into a stack. “Pop” is the term used to delete an element from the stack.**

**All insertions and deletions take place at the same end, so the last element added to the stack will be the first element removed from the stack. When a stack is created, the stack base remains fixed while the stack top changes as elements are added and removed. The most accessible element is the top and the least accessible element is the bottom of the stack.**

**REPRESENTATION OF STACK:**

**Let us consider a stack with 6 elements capacity. This is called as the size of the stack. The number of elements to be added should not exceed the maximum size of the stack. If we attempt to add new element beyond the maximum size, we will encounter a *stack overflow* condition. Similarly, you cannot remove elements beyond the base of the stack. If such is the case, we will reach a *stack underflow* condition.**

**When an element is added to a stack, the operation is performed by push(). Figure 3.1 shows the creation of a stack and addition of elements using push().**



***FIGURE 3.1: PUSH OPERATIONS ON STACK***

**When an element is taken off from the stack, the operation is performed by pop(). Figure 3.2 shows a stack initially with three elements and shows the deletion of elements using pop().**



***FIGURE 3.2: POP OPERATIONS ON STACK***

**APPLICATIONS OF STACK:**

* **Reversing a string of characters.**
* **Generating 3-address code from Polish postfix (or prefix) expressions.**
* **Handling function calls and returns, and recursion.**

**Arithmetic Expressions Polish Notation:**

**What is Polish Notation?**

**Conventionally, we use the operator symbol between its two operands in an arithmetic expression.**

**A+B C–D\*E A\*(B+C)**

**We can use parentheses to change the precedence of the operators. Operator precedence is pre-defined. This notation is called *INFIX notation*. Parentheses can change the precedence of evaluation. So Multiple are passes required for evaluation.**

**Polish notation:**

**It is named after Polish mathematician Jan Lukasiewicz.**

**Polish POSTFIX notation:**

**It refers to the notation in which the operator symbol is placed after its two operands.**

**AB+ CD\* AB\*CD+/**

**Polish PREFIX notation: It refers to the notation in which the operator symbol is placed before its two operands.**

**+AB \*CD /\*AB-CD**

**Advantages:**

**– No concept of operator priority.**

**• Simplifies the expression evaluation rule.**

**– No need of any parenthesis.**

**• Hence no ambiguity in the order of evaluation.**

**– Evaluation can be carried out using a single scan over the expression string.**

**• Using stack.**

**Converting an INFIX expression to POSTFIX:**

**The Algorithm (Q:: given infix expression, P:: output postfix expression)**

**push (‘(’);**

**Add “)” to the end of Q;**

**while (not end of string in Q do)**

**{**

**a = get\_next\_token();**

**if (a is an operand) add it to P;**

**if (a is ‘(’) push(a);**

**if (a is an operator)**

**{**

**Repeatedly pop from stack and add to P each operator (on top of the stack) which has the same or higher precedence than “a”;**

**push(a);**

**}**

**if (a is ‘)’)**

**{**

**Repeatedly p y pop from stack and add to P each operator (on the top of the stack) until a left parenthesis is encountered;**

**Remove the left parenthesis;**

**}**

**}**

**EXAMPLE:**







**PROGRAM-1: WRITE A PROGRAM TO SHOW PUSH AND POP FUNCTION OF STACK.**

**#include <stdio.h>**

**#include <stdlib.h> // For using the exit() function**

**#define max 50**

**void push();**

**void pop();**

**void display();**

**int menu();**

**int stack[max], top = 0;**

**int main() {**

**int ch;**

**do {**

**ch = menu();**

**switch (ch) {**

**case 1:**

**push();**

**break;**

**case 2:**

**pop();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0); // Use exit(0) to exit the program successfully**

**default:**

**printf("\nEnter a valid choice!!");**

**}**

**} while (1);**

**}**

**int menu() {**

**int ch;**

**printf("\nStack");**

**printf("\n1. Push\n2. Pop\n3. Display\n4. Exit");**

**printf("\nEnter your Choice: ");**

**scanf("%d", &ch);**

**return ch;**

**}**

**void push() {**

**if (top == max)**

**printf("\nOverflow");**

**else {**

**int element;**

**printf("\nEnter Element: ");**

**scanf("%d", &element);**

**printf("\nElement(%d) has been pushed at %d", element, top);**

**stack[top++] = element;**

**}**

**}**

**void pop() {**

**if (top == 0) // Correct condition for checking empty stack**

**printf("\nUnderflow");**

**else {**

**top--;**

**printf("\nElement has been popped out!");**

**}**

**}**

**void display() {**

**if (top == 0)**

**printf("\nStack is Empty!!");**

**else {**

**int i;**

**for (i = 0; i < top; i++) // Display only elements up to the current top**

**printf("%d ", stack[i]);**

**}**

**}**

**OUTPUT:**

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 1

Enter Element: 10

Element(10) has been pushed at 0

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 1

Enter Element: 20

Element(20) has been pushed at 1

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 3

10 20

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 1

Enter Element: 30

Element(30) has been pushed at 2

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 2

Element has been popped out!

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 3

10 20

Stack

1. Push

2. Pop

3. Display

4. Exit

Enter your Choice: 4

**PROGRAM-2: WRITE A PROGRAM TO CONVERT AN INFIX TO POSTFIX EXPRESSION:**

**#include <stdio.h>**

**#include <conio.h>**

**#include <string.h>**

**char postfix[50];**

**char infix[50];**

**char opstack[50]; /\* operator stack \*/**

**int i, j, top = 0;**

**int lesspriority(char op, char op\_at\_stack) {**

**int k;**

**int pv1; /\* priority value of op \*/**

**int pv2; /\* priority value of op\_at\_stack \*/**

**char operators[] = {'+', '-', '\*', '/', '%', '^', '('};**

**int priority\_value[] = {0, 0, 1, 1, 2, 3, 4};**

**if (op\_at\_stack == '(')**

**return 0;**

**for (k = 0; k < 6; k++) {**

**if (op == operators[k])**

**pv1 = priority\_value[k];**

**}**

**for (k = 0; k < 6; k++) {**

**if (op\_at\_stack == operators[k])**

**pv2 = priority\_value[k];**

**}**

**if (pv1 < pv2)**

**return 1;**

**else**

**return 0;**

**}**

**void push(char op) /\* op - operator \*/ {**

**/\* before pushing the operator 'op' into the stack, check priority**

**of op with top of opstack if less, then pop the operator from stack**

**then push into postfix string else push op onto stack itself \*/**

**if (top == 0) {**

**opstack[top] = op;**

**top++;**

**} else {**

**if (op != '(') {**

**while (lesspriority(op, opstack[top - 1]) == 1 && top > 0) {**

**postfix[j] = opstack[--top];**

**j++;**

**}**

**}**

**opstack[top] = op; /\* pushing onto stack \*/**

**top++;**

**}**

**}**

**void pop() {**

**while (opstack[--top] != '(') /\* pop until '(' comes \*/ {**

**postfix[j] = opstack[top];**

**j++;**

**}**

**}**

**int main() {**

**char ch;**

**clrscr();**

**printf("\n Enter Infix Expression : ");**

**gets(infix);**

**while ((ch = infix[i++]) != '\0') {**

**switch (ch) {**

**case ' ':**

**break;**

**case '(':**

**case '+':**

**case '-':**

**case '\*':**

**case '/':**

**case '^':**

**case '%':**

**push(ch); /\* check priority and push \*/**

**break;**

**case ')':**

**pop();**

**break;**

**default:**

**postfix[j] = ch;**

**j++;**

**}**

**}**

**while (top >= 0) {**

**postfix[j] = opstack[--top];**

**j++;**

**}**

**postfix[j] = '\0';**

**printf("\n Infix Expression : %s ", infix);**

**printf("\n Postfix Expression : %s ", postfix);**

**getch();**

**return 0;**

**}**

**OUTPUT:**

**Enter Infix Expression : (5 + 3) \* 4 - 2**

**Infix Expression : (5 + 3) \* 4 - 2**

**Postfix Expression : 5 3 + 4 \* 2 -**