**ASHESH KUMAR**

**501254**

**I.T. – III**

**STATIC IMPLEMENTATION OF STACK OPERATIONS & APPLICATION**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <time.h> //required to delay execution, :)

/\* TO CONVERTE LARGE EXPRESSIONS CHANGE N\*/

#define N 9

#define LP 10

#define RP 20

#define OPERATOR 30

#define OPERAND 40

// Minimum Left parentheses precedence

#define LPP 0

// + & -

#define AP 1

#define SP AP

// \* & /

#define MP 2

#define DP MP

// Remainder precedence.

#define REMP 2

#define NONE 9

/\*Function to Push\*/

void push(int element);

/\*Function to Pop\*/

int pop(void);

/\*Function to Display stack\*/

void display(void);

/\*CONVERSION FUNCTION\*/

void infixtopostfix(void);

/\*TYPE OF INPUT CHARACTER\*/

int gettype(char);

/\*PRECEDENCE CHECKER\*/

int getprec(char);

/\*Global declarations\*/

static int stackar[N], top = -1;

static char infix[N + 1], postfix[N + 1];

int main() {

char choice, ch;

int n, po; //this is local to main()

do {

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("A.PUSH\nB.POP\nC.INFIX TO POSTFIX & EVAL\nD.DISPLAY\nE.EXIT\n");

printf("GIVE CHOICE: ");

choice = getchar();

fflush(stdin); //clear any extra characters or whitespaces typed in after "choice"

printf("\n");

switch (choice) {

case 'A':

case 'a':

printf("push number?: ");

scanf("%d", &n);

fflush(stdin);

push(n);

if (top != (N - 1))

printf("%d PUSHED!\n\n", stackar[top]);

break;

case 'B':

case 'b':

po = pop();

if (top != -1)

printf("%d POPED!\n\n", po);

break;

case 'C':

case 'c':

top = -1;

printf("\n\nEnter infix expression:");

gets(infix);

infixtopostfix();

printf("\nFilling the stack...wait!\n");

sleep(2); //Delays the execution

printf("Stack filled!\n");

printf("\ninfix = %s\npost fix =%s\n", infix, postfix);

printf("Evaluate the expression?[Y/N]: ");

ch = getche();

if (ch == 'Y' || ch == 'y') {

int i = 0, op1, op2;

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

if (isdigit(ch))

push(ch - '0'); //Push the operand

else {

/\*Operator,pop two operands\*/

op2 = pop();

op1 = pop();

switch (ch) {

case '+':push(op1 + op2);

break;

case '-':push(op1 - op2);

break;

case '\*':push(op1 \* op2);

break;

case '/':push(op1 / op2);

break;

case '%':push(op1 % op2);

break;

}

}

}

printf("\n Given Postfix Expn: %s\n", postfix);

printf("\n Result after Evaluation: %d\n", stackar[top]);

top = -1; //confirms that stackar[] is empty

} else

break;

printf("\nStack has now been emptied,Ready\n");

break;

case 'D':

case 'd':

display();

break;

case 'E':

case 'e':

printf("PROGRAM WILL EXIT\_\_");

exit(0); //exits all loops

default:

printf("INCORRECT CHOICE! TRY AGAIN!!\n\n");

}

} while (choice != 'e' || choice != 'E');

return (0);

}

void push(int element) {

if (top >= N - 1)

printf("\nOPERATION FAILED.\nTHE STACK IS FULL!!\n\n");

else {

++top;

stackar[top] = element;

}

}

int pop() {

if (top <= -1)

printf("\nOPERATION FAILED.\nTHE STACK IS EMPTY!!\n\n");

else {

return (stackar[top--]);

}

}

void display() {

int i;

printf("\n#THE STACK#\n\n");

/\*shows empty\*/

for (i = 0; i < N - (top + 1); i++) {

printf(" | |\n");

printf(" \_\_\_ \n");

}

/\*shows elements\*/

for (i = 0; i <= top; i++) {

printf(" |%3d|", stackar[top - i]);

if (i == 0) printf("<--TOP");

printf("\n");

printf(" \_\_\_ \n");

}

}

void infixtopostfix(void) {

int i, p, l, type, prec;

char next;

i = p = 0;

l = strlen(infix);

while (i < l) {

type = gettype(infix[i]);

switch (type) {

case LP:

push(infix[i]);

break;

case RP:

while ((next = pop()) != '(')

postfix[p++] = next;

break;

case OPERAND:

postfix[p++] = infix[i];

break;

case OPERATOR:

prec = getprec(infix[i]);

//typecasting in while loop, declars "stackar[top]" as char.

while (top>-1 && prec <= getprec((char) (stackar[top]))) {

postfix[p++] = pop();

}

push(infix[i]);

break;

}

i++;

}

while (top>-1)

postfix[p++] = pop();

postfix[p] = '\0';

}

int gettype(char sym) {

switch (sym) {

case '(':

return (LP);

case ')':

return (RP);

case '+':

case '-':

case '\*':

case '/':

case '%':

return (OPERATOR);

default:

return (OPERAND);

}

}

int getprec(char sym) {

switch (sym) {

case '(':

return (LPP);

case '+':

return (AP);

case '-':

return (SP);

case '\*':

return (MP);

case '/':

return (DP);

case '%':

return (REMP);

default:

return (NONE);

}

}

/\*\*

\*NOTE:

\*1)printf() for case B doesn't work when emptying the last element

\*2)use numbers within the range of int data type

\*3)evaluation of postfix works ONLY for SINGLE DIGITS

\*/

**OUTPUT**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: a

push number?: 1

1 PUSHED!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: a

push number?: 13

13 PUSHED!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: d

#THE STACK#

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| 13|<--TOP

\_\_\_

| 1|

\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: b

13 POPED!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: d

#THE STACK#

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| |

\_\_\_

| 1|<--TOP

\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: c

Enter infix expression:(4-7)\*6/9

Filling the stack...wait!

Stack filled!

infix = (4-7)\*6/9

post fix =47-6\*9/

Evaluate the expression?[Y/N]: y

Given Postfix Expn: 47-6\*9/

Result after Evaluation: -2

Stack has now been emptied,Ready

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A.PUSH

B.POP

C.INFIX TO POSTFIX & EVAL

D.DISPLAY

E.EXIT

GIVE CHOICE: E

PROGRAM WILL EXIT\_\_