**Stack Linked List**

#include<stdio.h>

#include<conio.h>

struct Node

{

int data;

struct Node \*next;

}\*top = NULL;

void push(int);

void pop();

void display();

main()

{

int choice, value;

printf("\n:: Stack using Linked List ::\n");

while(1){

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

printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d", &value);

push(value);

break;

case 2: pop(); break;

case 3: display(); break;

case 4: break;

default: printf("\nWrong selection!!! Please try again!!!\n");

}

}

}

void push(int value)

{

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

if(top == NULL)

newNode->next = NULL;

else

newNode->next = top;

top = newNode;

printf("\nInsertion is Success!!!\n");

}

void pop()

{

if(top == NULL)

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

else{

struct Node \*temp = top;

printf("\nDeleted element: %d", temp->data);

top = temp->next;

free(temp);

}

}

void display()

{

if(top == NULL)

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

else{

struct Node \*temp = top;

while(temp->next != NULL){

printf("%d--->",temp->data);

temp = temp -> next;

}

printf("%d--->NULL",temp->data);

}

}

**Evaluation of Expression**

#include <stdio.h>

#include <ctype.h>

#define MAXSTACK 100

#define POSTFIXSIZE 100

int stack[MAXSTACK];

int top = -1;

void push(int item)

{

if (top >= MAXSTACK - 1)

{

printf("stack over flow");

return;

}

else

{

top = top + 1;

stack[top] = item;

}

}

int pop()

{

int item;

if (top < 0)

{

printf("stack under flow");

}

else

{

item = stack[top];

top = top - 1;

return item;

}

}

voidEvalPostfix(char postfix[])

{

inti;

charch;

intval;

int A,B;

for (i = 0; postfix[i] != ')'; i++)

{

ch = postfix[i];

if (isdigit(ch))

{

push(ch - '0');

}

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/')

{

A = pop();

B = pop();

switch (ch)

{

case '\*':

val = B \* A;

break;

case '/':

val = B / A;

break;

case '+':

val = B + A;

break;

case '-':

val = B - A;

break;

}

push(val);

}

}

if(top<=0)

printf(" \n Result of expression evaluation : %d \n", pop());

else

printf("incomplete expression");

}

int main()

{

inti;

char postfix[POSTFIXSIZE];

printf("ASSUMPTION: There are only four operators(\*, /, +, -) in an expression and operand is single digit only.\n");

printf(" \nEnter postfix expression,\npress right parenthesis ')' for end expression : ");

for (i = 0; i<= POSTFIXSIZE - 1; i++)

{

scanf("%c", &postfix[i]);

if (postfix[i] == ')')

{

break;

}

}

EvalPostfix(postfix);

return 0;

}

**Infix to postfix**

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define SIZE 100

char stack[SIZE];

int top = -1;

void push(char item)

{

if(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

intis\_operator(char symbol)

{

if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

return 1;

}

else

{

return 0;

}

}

int precedence(char symbol)

{

if(symbol == '^')/\* exponent operator, highest precedence\*/

{

return(3);

}

else if(symbol == '\*' || symbol == '/')

{

return(2);

}

else if(symbol == '+' || symbol == '-') /\* lowest precedence \*/

{

return(1);

}

else

{

return(0);

}

}

voidInfixToPostfix(char infix\_exp[], char postfix\_exp[])

{

inti, j;

char item;

char x;

push('(');

strcat(infix\_exp,")");

i=0;

j=0;

item=infix\_exp[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item;

j++;

}

else if(is\_operator(item) == 1) /\* means symbol is operator \*/

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x; /\* so pop all higher precendence operator and \*/

j++;

x = pop(); /\* add them to postfix expresion \*/

}

push(x);

/\* because just above while loop will terminate we have

oppped one extra item

for which condition fails and loop terminates, so that one\*/

push(item); /\* push current oprerator symbol onto stack \*/

}

else if(item == ')') /\* if current symbol is ')' then \*/

{

x = pop(); /\* pop and keep popping until \*/

while(x != '(') /\* '(' encounterd \*/

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

else

{ /\* if current symbol is neither operand not '(' nor ')' and nor

operator \*/

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

i++;

item = infix\_exp[i]; /\* go to next symbol of infix expression \*/

} /\* while loop ends here \*/

if(top>0)

{

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

postfix\_exp[j] = '\0'; /\* add sentinel else puts() fucntion \*/

/\* will print entire postfix[] array upto SIZE \*/

}

/\* main function begins \*/

int main()

{

char infix[SIZE], postfix[SIZE]; /\* declare infix string and postfix string \*/

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix); /\* call to convert \*/

printf("Postfix Expression: ");

puts(postfix); /\* print postfix expression \*/

return 0;

}