Array implementation of stack

#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

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

printf("\n Press Next Choice");

}

else

{

printf("\n The STACK is empty");

}

}

Linked list implementation of stack

1. #include <stdio.h>
2. #include <stdlib.h>
3. **void** push();
4. **void** pop();
5. **void** display();
6. struct node
7. {
8. **int** val;
9. struct node \*next;
10. };
11. struct node \*head;
13. **void** main ()
14. {
15. **int** choice=0;
16. printf("\n\*\*\*\*\*\*\*\*\*Stack operations using linked list\*\*\*\*\*\*\*\*\*\n");
17. printf("\n----------------------------------------------\n");
18. **while**(choice != 4)
19. {
20. printf("\n\nChose one from the below options...\n");
21. printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
22. printf("\n Enter your choice \n");
23. scanf("%d",&choice);
24. **switch**(choice)
25. {
26. **case** 1:
27. {
28. push();
29. **break**;
30. }
31. **case** 2:
32. {
33. pop();
34. **break**;
35. }
36. **case** 3:
37. {
38. display();
39. **break**;
40. }
41. **case** 4:
42. {
43. printf("Exiting....");
44. **break**;
45. }
46. **default**:
47. {
48. printf("Please Enter valid choice ");
49. }
50. };
51. }
52. }
53. **void** push ()
54. {
55. **int** val;
56. struct node \*ptr = (struct node\*)malloc(sizeof(struct node));
57. **if**(ptr == NULL)
58. {
59. printf("not able to push the element");
60. }
61. **else**
62. {
63. printf("Enter the value");
64. scanf("%d",&val);
65. **if**(head==NULL)
66. {
67. ptr->val = val;
68. ptr -> next = NULL;
69. head=ptr;
70. }
71. **else**
72. {
73. ptr->val = val;
74. ptr->next = head;
75. head=ptr;
77. }
78. printf("Item pushed");
80. }
81. }
83. **void** pop()
84. {
85. **int** item;
86. struct node \*ptr;
87. **if** (head == NULL)
88. {
89. printf("Underflow");
90. }
91. **else**
92. {
93. item = head->val;
94. ptr = head;
95. head = head->next;
96. free(ptr);
97. printf("Item popped");
99. }
100. }
101. **void** display()
102. {
103. **int** i;
104. struct node \*ptr;
105. ptr=head;
106. **if**(ptr == NULL)
107. {
108. printf("Stack is empty\n");
109. }
110. **else**
111. {
112. printf("Printing Stack elements \n");
113. **while**(ptr!=NULL)
114. {
115. printf("%d\n",ptr->val);
116. ptr = ptr->next;
117. }
118. }
119. }

/\* POLYNOMIAL ADDITION AND SUBTRACTION \*/

#include<stdio.h>

#include<conio.h>

#include<malloc.h>

struct node

{

int coeff,pow;

struct node \*next;

}\*start1=NULL,\*start2=NULL,\*start=NULL,\*start3=NULL,\*p,\*p1,\*p2,\*temp,\*a=NULL;

void create(struct node \*);

void display(struct node \*);

void add(struct node \*,struct node \*,struct node \*);

void sub(struct node \*,struct node \*,struct node \*);

void main()

{

clrscr();

start1=(struct node \*)malloc(sizeof(struct node));

start2=(struct node \*)malloc(sizeof(struct node));

start3=(struct node \*)malloc(sizeof(struct node));

printf("\nEnter 1st Polynomial\n");

create(start1);

printf("\nEnter 2nd Polynomial\n");

create(start2);

printf("\n1st Polynomial\n");

display(start1);

printf("\n2nd Polynomial\n");

display(start2);

add(start1,start2,start3);

printf("\nAdded Polynomial: \n");

display(start3);

start3=NULL;

sub(start1,start2,start3);

printf("\nSubtracted Polynomial: \n");

display(start3);

getch();

}

//-------CREATE-------

void create(struct node \*temp)

{

char ch;

do

{

printf("Enter coefficient and power\n ");

scanf("%d%d",&temp->coeff,&temp->pow);

temp->next=(struct node \*)malloc(sizeof(struct node));

temp=temp->next;

temp->next=NULL;

printf("Press y to continue\n");

ch=getch(); //To add terms to the polynomial

}while(ch=='y'|| ch=='Y');

}

//-------DISPLAY-------

void display(struct node \*start)

{

p=start;

while(p->next!=NULL)

{

printf("%dx^%d",p->coeff,p->pow);

p=p->next;

if(p->next!=NULL)

printf(" + ");

}

printf("\n");

}

//-------ADDITION-------

void add(struct node \*start1,struct node \*start2,struct node \*start3)

{

while(start1->next && start2->next)

{

if(start1->pow==start2->pow)

{

start3->coeff=start1->coeff+start2->coeff;

start3->pow=start1->pow;

start1=start1->next;

start2=start2->next;

}

else if(start1->pow > start2->pow)

{

start3->coeff=start1->coeff;

start3->pow=start1->pow;

start1=start1->next;

}

else

{

start3->coeff=start2->coeff;

start3->pow=start2->pow;

start2=start2->next;

}

start3->next=(struct node \*)malloc(sizeof (struct node));

start3=start3->next;

start3->next=NULL;

}

while(start1->next || start2->next)

{

if(start1->next)

{

start3->pow=start1->pow;

start3->coeff=start1->coeff;

start1=start1->next;

}

if(start2->next)

{

start3->coeff=start2->coeff;

start3->pow=start2->pow;

start2=start2->next;

}

start3->next=(struct node \*)malloc(sizeof(struct node));

start3=start3->next;

start3->next=NULL;

}

}

//-------SUBTRACTION-------

void sub(struct node \*start1,struct node \*start2,struct node \*start3)

{

while(start1->next && start2->next)

{

if(start1->pow==start2->pow)

{

start3->coeff=start1->coeff-start2->coeff;

start3->pow=start1->pow;

start1=start1->next;

start2=start2->next;

}

else if(start1->pow > start2->pow)

{

start3->coeff=start1->coeff;

start3->pow=start1->pow;

start1=start1->next;

}

else

{

start3->coeff=-start2->coeff;

start3->pow=start2->pow;

start2=start2->next;

}

start3->next=(struct node \*)malloc(sizeof (struct node));

start3=start3->next;

start3->next=NULL;

}

while(start1->next || start2->next)

{

if(start1->next)

{

start3->pow=start1->pow;

start3->coeff=start1->coeff;

start1=start1->next;

}

if(start2->next)

{

start3->coeff=start2->coeff;

start3->pow=start2->pow;

start2=start2->next;

}

start3->next=(struct node \*)malloc(sizeof(struct node));

start3=start3->next;

start3->next=NULL;

}

}

**C-Program to Convert Infix to Prefix Expression**

|  |
| --- |
| /\* This program converts infix expression to prefix expression.   \* This program assume that there are Five operators: (\*, /, +, -,^)   \* This program will not work for fractional numbers.   \* Further this program does not check whether infix expression is valid or not in terms of number of operators and operands.\*/ #include<stdio.h> // for exit() function #include<stdlib.h> // for isdigit(char ) function #include<ctype.h> #include<string.h> #define SIZE 100  // Global Variable Declaration char stack[SIZE]; int top = -1;  //Global Function Declaration void push(char c); char pop(); int isoperator(char symbol); int precedence(char symbol); void InfixToPrefix(char infix\_exp[], char prefix\_exp[]);  // main() function begins void main() { // Declare infix string and prefix string char infix[SIZE], prefix[SIZE]; printf("\n\n Enter Infix expression : "); gets(infix); // Call to convert InfixToPrefix(infix,prefix); printf("\n Prefix Expression: "); // Print prefix expression puts(prefix); }  void InfixToPrefix(char infix\_exp[], char prefix\_exp[]) { int i, j, k, pos, len; char item, x, rev[SIZE]; // Reverse the infix expression pos=0; len=strlen(infix\_exp); for(k=len-1;k>=0;k--) { rev[pos]=infix\_exp[k]; pos++; } rev[pos]='\0'; strcpy(infix\_exp,rev); // Make Every “ ( ” as “ ) ” and every “ ) ” as “ ( ” for(i=0; infix\_exp[i]!='\0'; i++) { if(infix\_exp[i] == ')') infix\_exp[i] = '('; else if(infix\_exp[i] == '(') infix\_exp[i] = ')'; } //Convert expression to postfix form. // push '(' onto stack push('('); // add ')' to infix expression strcat(infix\_exp,")"); i=0; j=0; // Initialize before loop item=infix\_exp[i]; // Run loop till end of infix expression while(item != '\0') { if(item == '(') { push(item); } else if( isdigit(item) || isalpha(item)) { // Add operand symbol to postfix expression prefix\_exp[j] = item; j++; } else if(isoperator(item) == 1) { // pop all higher precendence operator and add them to postfix expresion x=pop(); while(isoperator(x) == 1 && precedence(x)>= precedence(item)) { prefix\_exp[j] = x; j++; x = pop(); } // push the last pop oprerator symbol onto stack push(x); // push current oprerator symbol onto stack push(item); } // if current symbol is ')' then pop and keep popping until '(' encounterd else if(item == ')') { x = pop(); while(x != '(') { prefix\_exp[j] = x; j++; x = pop(); } } else { // if current symbol is neither operand not '(' nor ')' and nor operator printf("\nInvalid infix Expression.\n"); break; } i++; // Go to next symbol of infix expression item = infix\_exp[i]; } //End while loop if(top > 0) printf("\n Invalid infix Expression."); prefix\_exp[j] = '\0'; // Reverse the prefix expression. pos=0; len=strlen(prefix\_exp); for(k=len-1;k>=0;k--) { rev[pos]=prefix\_exp[k]; pos++; } rev[pos]='\0'; strcpy(prefix\_exp,rev); }  // Define push operation void push(char c) { if(top >= SIZE-1) printf("\n Stack Overflow."); else { top++; stack[top] = c; } }  // Define pop operation char pop() { char c; c='\0'; if(top < 0) printf("\n Stack Underflow."); else { c = stack[top]; top--; } return c; }  // Define function that is used to determine whether any symbol is operator or not int isoperator(char symbol) { if(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol == '-') return 1; else return 0; }  // Define fucntion that is used to assign precendence to operator. // In this fucntion we assume that higher integer value means higher precendence int precedence(char symbol) { if(symbol == '^') return(5); else if(symbol == '/') return(4); else if(symbol == '\*') return(3); else if(symbol == '+') return(2); else if(symbol == '-') return(1); else return(0); }  Infix to post fix program |

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c ",pop());

}return 0;

}