Program 1: Implementation of stacks using arrays

#include<stdio.h>

#include<stdlib.h>

#define MAX 5

int top=-1;

int stack[MAX];

void push(int);

int pop();

int peek();

void display();

int isfull();

int isempty();

int main()

{

int c,x;

do

{

printf("Select your choice:\n");

printf("1.Push 2.Pop 3.Display top 4.Display all 0.Exit\n");

scanf("%d",&c);

switch(c)

{

case 1:printf("\nEnter element to be pushed\n");

scanf("%d",&x);

push(x);

printf("\n");

break;

case 2:x=pop();

printf("%d was popped\n",x);

printf("\n");

break;

case 3:x=peek();

printf("Element at the top is %d\n",x);

printf("\n");

break;

case 4:display();

printf("\n");

break;

default:if(c)

printf("Invalid option. Select again\n");

}

}while(c!=0);

}

void push(int x)

{

if(isfull())

printf("STACK OVERFLOW\n");

else

stack[++top]=x;

}

int pop()

{

if(isempty())

{

printf("STACK UNDERFLOW\n");

exit(1);

}

else

return stack[top--];

}

int peek()

{

if(isempty())

{

printf("STACK UNDERFLOW\n");

exit(1);

}

else

return stack[top];

}

void display()

{

if(isempty())

printf("STACK UNDERFLOW\n");

else

{

int i=top;

printf("Elements in the stack\n");

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

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

printf("\n");

}

}

int isempty()

{

if(top==-1)

return 1;

else

return 0;

}

int isfull()

{

if(top==MAX-1)

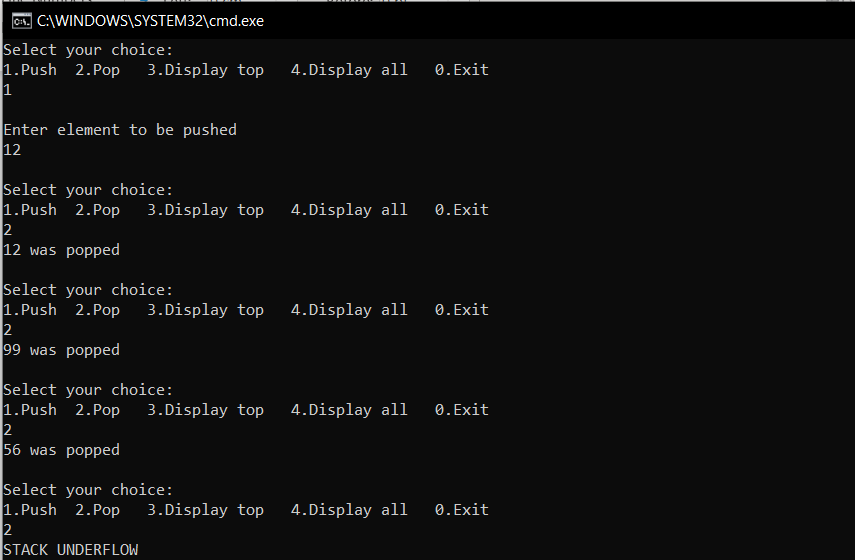
return 1;

else

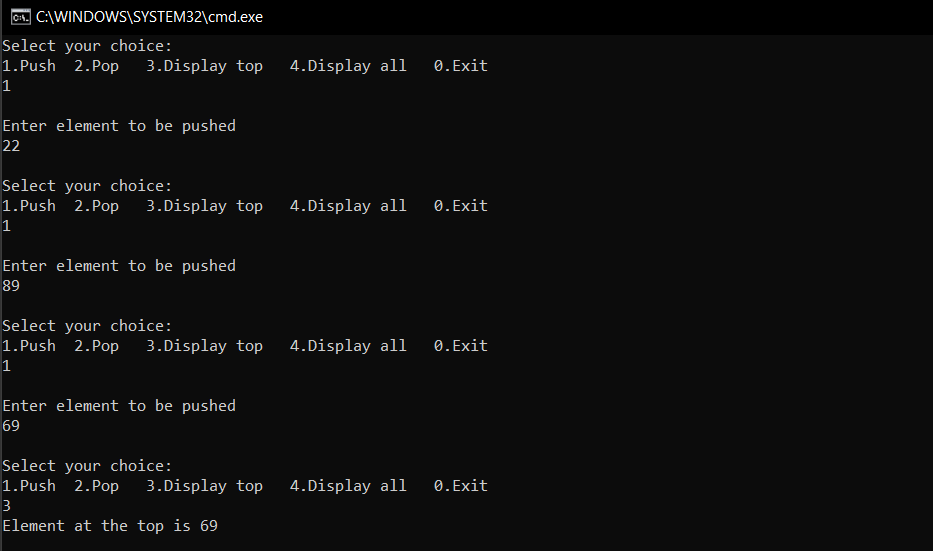
return 0;

}

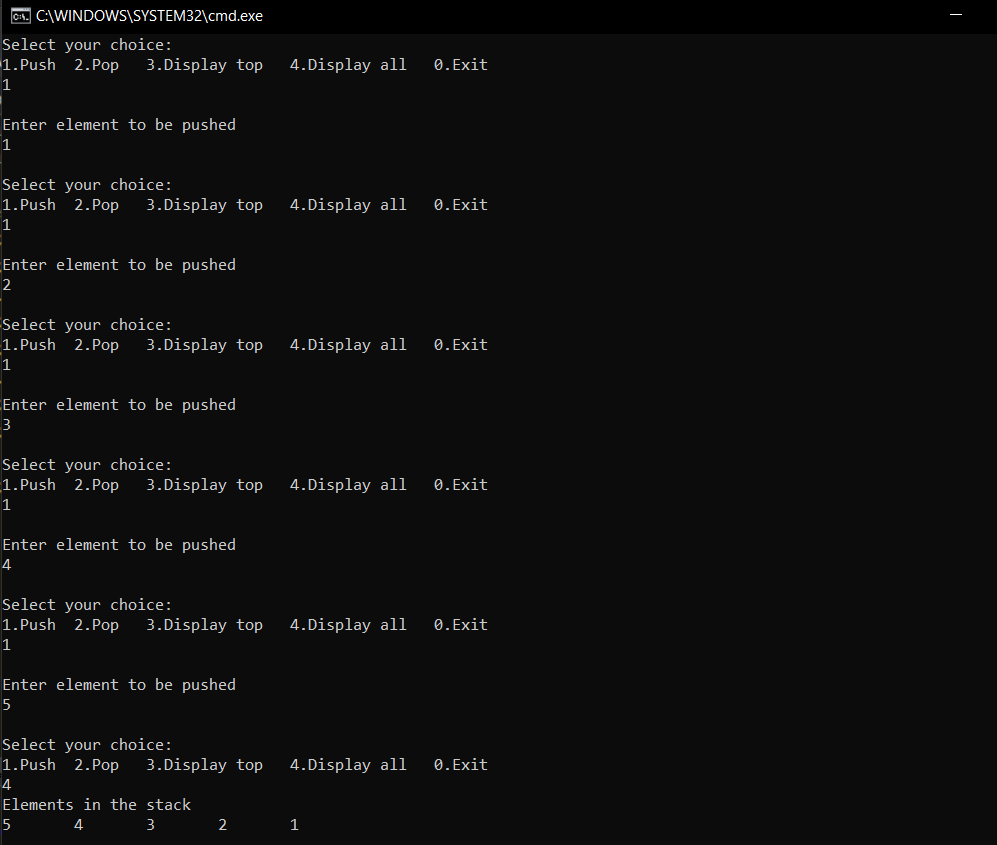
//output 1: push and pop until empty



//output 2: display top element



//output 3: display all elements



Program 2: Implementation of stacks using linked lists

#include<stdio.h>

#include<stdlib.h>

struct element

{

int data;

struct element\*next;

};

typedef struct element node;

int i=0;

void push(int);

int pop();

int peek();

void display();

int isfull();

int isempty();

node \*top=NULL;

int n;

int main()

{

int c,x;

printf("Enter number of elements\n");

scanf("%d",&n);

do

{

if(n<=0)

{

printf("Invalid number\n");

exit(1);

}

printf("Select your choice:\n");

printf("1.Push 2.Pop 3.Display top 4.Display all 0.Exit\n");

scanf("%d",&c);

switch(c)

{

case 1:printf("\nEnter element to be pushed\n");

scanf("%d",&x);

push(x);

printf("\n");

break;

case 2:x=pop();

printf("%d was popped\n",x);

printf("\n");

break;

case 3:x=peek();

printf("Element at the top is %d\n",x);

printf("\n");

break;

case 4:display();

printf("\n\n");

break;

default:if(c)

printf("Invalid option. Select again\n");

}

}while(c!=0);

}

void push(int x)

{

if(isfull())

printf("STACK OVERLFOW\n");

else

{

node \*temp=(node\*)malloc(sizeof(node));

temp->data=x;

temp->next=NULL;

temp->next=top;

top=temp;

i++;

}

}

int pop()

{

if(isempty())

{

printf("STACK UNDERFLOW\n");

exit(1);

}

else

{

int x;

node \*temp=top;

top=top->next;

x=temp->data;

free(temp);

return x;

}

}

int peek()

{

if(isempty())

{

printf("STACK UNDERFLOW\n");

exit(1);

}

else

return top->data;

}

void display()

{

if(isempty())

{

printf("STACK UNDERFLOW\n");

exit(1);

}

else

{

node \*ptr=top;

printf("\nThe elements in the stack are:\n");

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

}

int isfull()

{

if(i==n)

return 1;

else

return 0;

}

int isempty()

{

if(top==NULL)

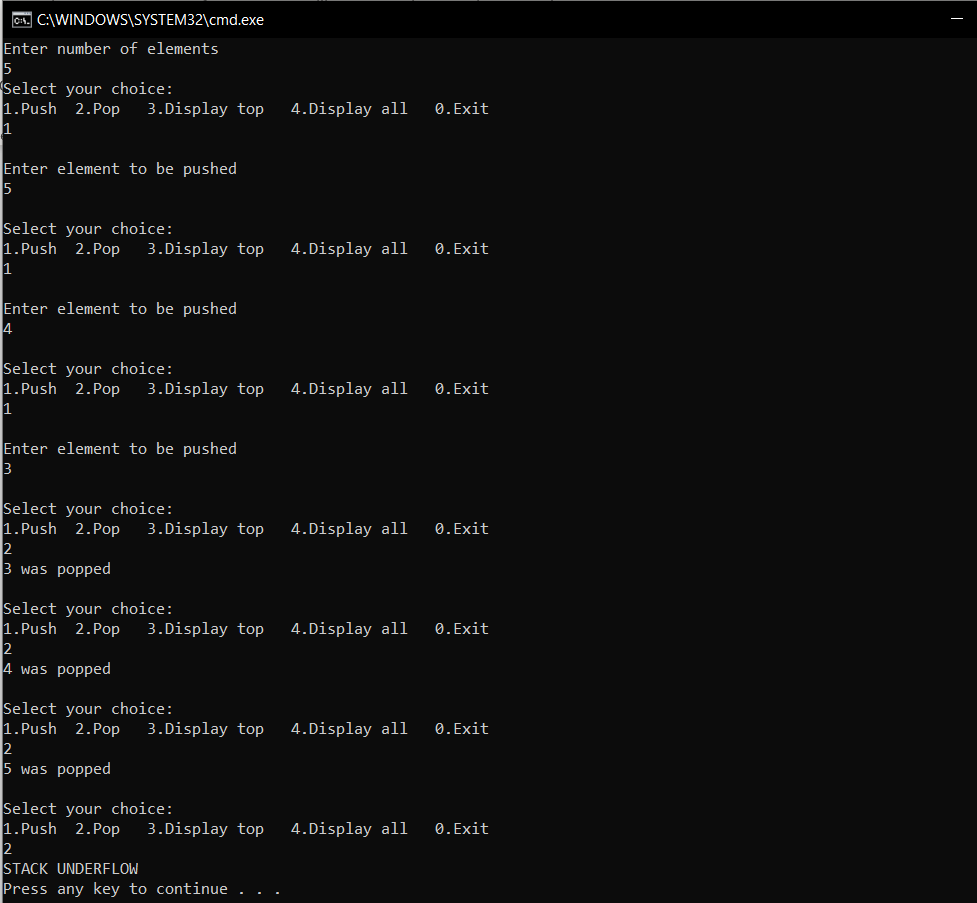
return 1;

else

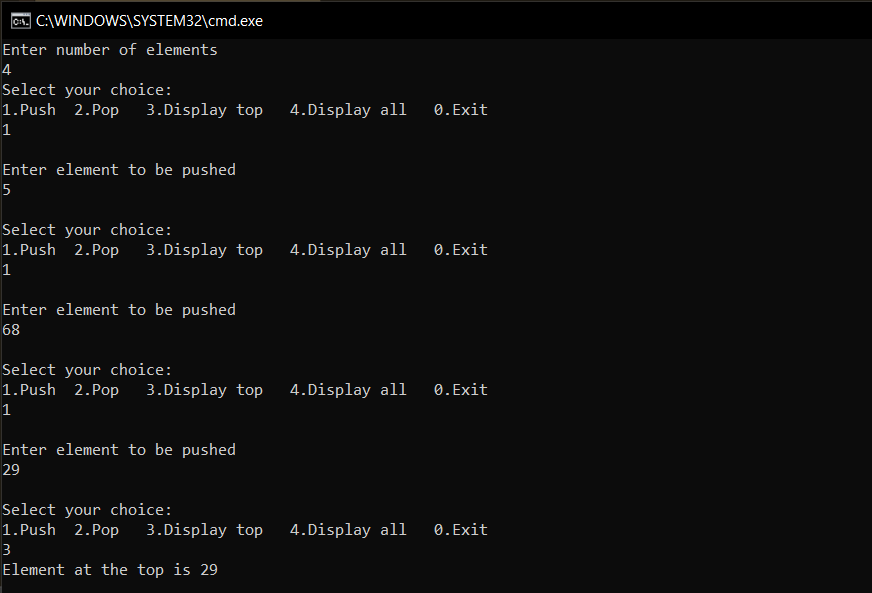
return 0;

}

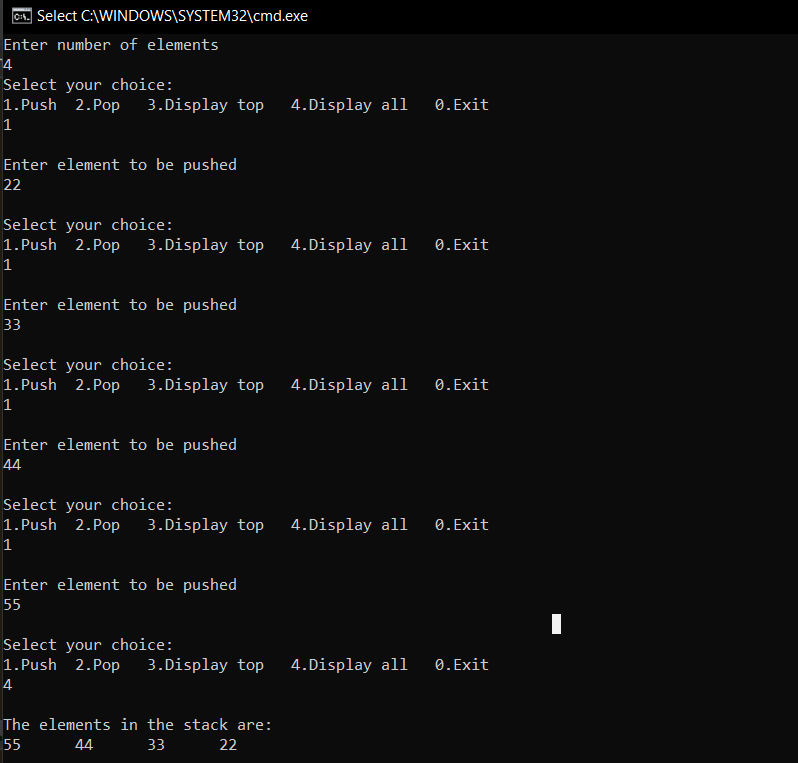
//output 1: push and pop until empty



//output 2: display top element



//output 3: display all elements



Program 3: Conversion of Infix expression to its Postfix equivalent

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#define TAB '\t'

#define SPACE ' '

struct element

{

float data;

struct element \*next;

}; typedef struct element node;

node \*top=NULL;

int p=-1;

char infix[30];

char postfix[30];

void push(float x);

int instack\_prio(char);

int incoming\_prio(char);

float pop();

int isempty();

void evaluate();

void display\_stack();

int main()

{

char x,z; int i;

printf("Enter your infix expression\n");

gets(infix);

for(i=0;i<strlen(infix);i++)

{

x=infix[i];

if(x==SPACE || x==TAB)

continue;

switch(x)

{

case '(': push(x);

break;

case '+':

case '-':

case '/':

case '\*':

case '^': while(!isempty() && incoming\_prio(x)<=instack\_prio(top->data))

{

z=pop();

postfix[++p]=z;

}

push(x);

//display\_stack();

break;

case ')': while( (z=pop())!='(')

postfix[++p]=z;

break;

default: postfix[++p]=x;

}

}

while(!isempty())

postfix[++p]=pop();

printf("The postfix equivalent is:");

puts(postfix);

evaluate();

}

int incoming\_prio(char x)

{

switch(x)

{

case '+':

case '-': return 1;

case '\*':

case '%':

case '/':return 2;

case '^':return 4;

default: return -1;

}

}

int instack\_prio(char x)

{

switch(x)

{

case '+':

case '-': return 1;

case '\*':

case '%':

case '/':return 2;

case '^':return 3;

default: return -1;

}

}

void push(float x)

{

node \*temp=(node\*)malloc(sizeof(node));

temp->next=NULL;

temp->data=x;

if(top==NULL)

top=temp;

else

{

temp->next=top;

top=temp;

}

}

float pop()

{

node \*temp=top; float x;

top=top->next;

x=temp->data;

free(temp);

return x;

}

int isempty()

{

if(top==NULL)

return 1;

else

return 0;

}

void evaluate()

{

float a,b,temp;

float val;

for(int i=0;i<=p;i++)

{

if(postfix[i]>='0' && postfix[i]<='9')

push(postfix[i]-'0');

else

{

a=pop();

b=pop();

switch(postfix[i])

{

case '+': temp=b+a;

break;

case '-':temp=b-a;

break;

case '\*':temp=b\*a;

break;

case '/':temp=b/a;

break;

case '%':temp=(int)b%(int)a;

break;

case '^':temp=pow(b,a);

break;

}

push((float)temp);

}

//display\_stack();

//printf("\n");

}

val=pop();

printf("The value of this expression is %f",val);

printf("\n\n");

}

void display\_stack()

{

node \*ptr=top;

while(ptr!=NULL)

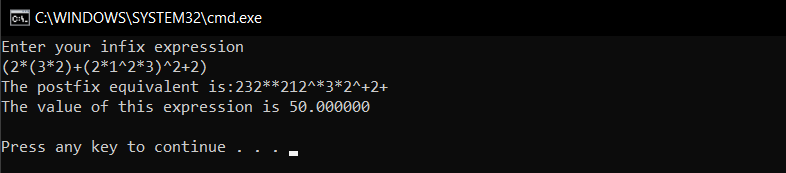
{

printf("%f",ptr->data);

ptr=ptr->next;

}

}



Program 4: Reversal of a string using stacks(LL)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct element

{

char data;

struct element\*next;

};

typedef struct element node;

node \*top=NULL;

void push(char);

char pop();

int main()

{

char a[30];

int n,i;

printf("Enter a string to reverse it\n");

gets(a);

n=strlen(a);

for(i=0;i<n;i++)

push(a[i]);

for(i=0;i<n;i++)

a[i]=pop();

printf("\nThe reversed string is:\n");

puts(a);

}

void push(char x)

{

node \*temp=(node\*)malloc(sizeof(node));

temp->data=x;

temp->next=NULL;

temp->next=top;

top=temp;

}

char pop()

{

node \*temp=top;

char x;

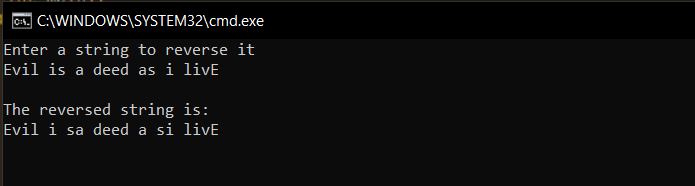
x=temp->data;

top=top->next;

free(temp);

return x;

}



Program 5: Nesting of Parentheses using stacks (LL)

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define SPACE ' '

#define TAB '\t'

struct element

{

char data;

struct element \*next;

};

typedef struct element node;

node \*top=NULL;

void push(char);

char pop();

int compare(char);

int main()

{

int i;

char a[30];

start:

printf("Enter your parentheses\n");

gets(a);

for(i=0;i<strlen(a);i++)

{ char c=a[i];

if(c!=SPACE && c!=TAB)

{

switch(c)

{

case '(':

case '[':

case '{': push(c);

break;

case ')':

case ']':

case '}':if(!compare(c))

{

printf("Parentheses are not balanced\n");

exit(1);

}

break;

default:printf("Please enter only brackets\n");

goto start;

}

} //[()]{}{[()()]()}

}

if(top!=NULL)

printf("Parentheses are not balanced\n");

else

printf("Parentheses are balanced\n");

}

int compare(char x)

{

switch(x)

{

case ')':if(pop()!='(')

return 0;

break;

case ']':if(pop()!='[')

return 0;

break;

case '}':if(pop()!='{')

return 0;

break;

default: return -1;

}

return 1;

}

void push(char x)

{

node \*temp=(node\*)malloc(sizeof(node));

temp->data=x;

temp->next=NULL;

temp->next=top;

top=temp;

}

char pop()

{

if(top==NULL)

{

printf("Parentheses are unbalanced\n");

exit(1);

}

else

{

char x;

node \*temp=top;

top=top->next;

x=temp->data;

free(temp);

return x;

}

}

