1.C programme to reverse a string using stack

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#define MAX 20

int top = -1;

char stack[MAX];

char pop();

void push(char);

int main()

{

char str[20];

unsigned int i;

printf("Enter the string : " );

gets(str);

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

push(str[i]);

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

str[i]=pop();

printf("\nReversed string is : ");

puts(str);

return 0;

}

void push(char item)

{

if(top == (MAX-1))

{

printf("\nStack Overflow\n");

return;

}

stack[++top] =item;

}/\*End of push()\*/

char pop()

{

if(top == -1)

{

printf("\nStack Underflow\n");

exit(1);

}

return stack[top--];

}

2.c programme converting infix to post fix

#include<stdio.h>  
#include<string.h>  
char stack[50];  
int top=-1;  
void post(char infix[]);  
void push(char);  
char pop();  
  
void main()  
{  
    char infix[25];  
    printf("\nENTER THE INFIX EXPRESSION = ");  
    gets(infix);  
    post(infix);  
    getch();  
}  
  
void push(char symb)  
{  
    if(top>=49)  
    {  
          printf("\nSTACK OVERFLOW");  
         getch();  
         return;  
    }  
    else  
    {  
          top=top+1;  
         stack[top]=symb;  
    }  
}  
char pop()  
{  
     char item;  
     if(top==-1)  
     {  
            printf("\nSTACK IS EMPTY");  
            getch();  
            return(0);  
      }  
      else  
     {  
            item=stack[top];  
            top--;  
     }  
     return(item);  
}  
int preced(char ch)  
{  
      if(ch==47)  
      {  
             return(5);  
      }  
      else if(ch==42)  
      {  
            return(4);  
      }  
      else if(ch==43)  
      {  
             return(3);  
      }  
      else  
      return(2);  
}  
void post(char infix[])  
{  
      int l;  
      int index=0,pos=0;  
      char symbol,temp;  
      char postfix[40];  
      l=strlen(infix);  
      push('#');  
      while(index<l)  
      {  
             symbol=infix[index];  
             switch(symbol)  
             {  
                    case '(': push(symbol);  
                    break;  
                    case ')': temp=pop();  
                    while(temp!='(')  
                    {  
                            postfix[pos]=temp;  
                            pos++;  
                            temp=pop();  
                    }  
                    break;  
                    case '+':  
                    case '-':  
                    case '\*':  
                    case '/':  
                    case '^':  
                    while(preced(stack[top])>=preced(symbol))  
                    {  
                            temp=pop();  
                            postfix[pos]=temp;  
                            pos++;  
                    }  
                    push(symbol);  
                    break;  
                    default: postfix[pos++]=symbol;  
                    break;  
            }  
            index++;  
      }  
      while(top>0)  
      {  
               temp=pop();  
               postfix[pos++]=temp;  
      }  
       postfix[pos++]='\0';  
       puts(postfix);  
       return;  
}

3.C programme Queues using stacks

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

struct Stack {

int data;

struct Stack \*next;

};

struct Stack \*CreateStack () {

return NULL;

}

int isEmptyStack(struct Stack \*top) {

return (top == NULL);

}

void Push(struct Stack \*\*top, int data) {

struct Stack newNode = (struct Stack) malloc(sizeof(struct Stack));

if(!newNode)

return;

newNode->data = data;

newNode->next = \*top;

\*top = newNode;

}

int Pop(struct Stack \*\*top) {

struct Stack \*temp;

int data;

if(isEmptyStack(\*top)) {

printf("Empty Stack.\n");

return INT\_MIN;

}

temp = \*top;

data = (\*top)->data;

\*top = (\*top)->next;

free(temp);

return data;

}

4.C programme to BJT

#​include​ ​<​stdio.h​>​

#​include​ ​<​stdlib.h​>​

​struct​ node {

​int​ data;

​struct​ node\* left;

​struct​ node\* right;

};

​struct​ node \*​newNode​(​int​ item)

{

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

temp->​data​ = item;

temp->​left​ = temp->​right​ = ​NULL​;

​return​ temp;

}

​struct​ node\* ​insert​(​struct​ node \*node, ​int​ value)

{

​if​ (node == ​NULL​) ​return​ ​newNode​(value);

​if​ (value < node->​data​)

node->​left​ = ​insert​(node->​left​, value);

​else​ ​if​ (value > node->​data​)

node->​right​ = ​insert​(node->​right​, value);

​return​ node;

}

​struct​ node \* ​minValueNode​(​struct​ node\* node)

{

​struct​ node\* current = node;

​while​ (current && current->​left​ != ​NULL​)

current = current->​left​;

​return​ current;

}

​struct​ node\* ​deleteNode​(​struct​ node\* root, ​int​ data)

{

​if​ (root == ​NULL​) ​return​ root;

​if​ (data < root->​data​)

root->​left​ = ​deleteNode​(root->​left​, data);

​else​ ​if​ (data > root->​data​)

root->​right​ = ​deleteNode​(root->​right​, data);

​else​

{

​if​ (root->​left​ == ​NULL​)

{

​struct​ node \*temp = root->​right​;

​free​(root);

​return​ temp;

}

​else​ ​if​ (root->​right​ == ​NULL​)

{

​struct​ node \*temp = root->​left​;

​free​(root);

​return​ temp;

}

​struct​ node\* temp = ​minValueNode​(root->​right​);

root->​data​ = temp->​data​;

root->​right​ = ​deleteNode​(root->​right​, temp->​data​);

}

​return​ root;

}

​void​ ​inorder​(​struct​ node\* root){

​if​(root == ​NULL​) ​return​;

​inorder​(root->​left​);

​printf​(​"​%d​ ->​"​, root->​data​);

​inorder​(root->​right​);

}

​void​ ​main​()

{

​struct​ node \*root = ​NULL​;

root = ​insert​(root, ​50​);

​insert​(root, ​1​);

​insert​(root, ​16​);

​insert​(root, ​22);

​

​printf​(​"​\n​BST before deleting 43\n​"​);

​inorder​(root);

​deleteNode​(root,​43​);

​printf​(​"​\n​BST after deletion​\n​"​);

​inorder​(root);

}