Program-07

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

#define MAX 20

void InfixtoPostfix(char source[],char target[]);

int getpriority(char);

void push(char [],char);

char pop(char []);

char st[MAX];

int top=-1;

int main()

{

char infix[20],postfix[20];

clrscr();

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

gets(infix);

InfixtoPostfix(infix,postfix);

printf("\n The Corresponding postfix expression is :");

puts(postfix);

getch();

return 0;

}

void InfixtoPostfix(char source[],char target[])

{

int i=0,j=0;

char temp;

strcpy(target," ");

while(source[i] !='\0')

{

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

{

push(st,source[i]);

i++;

}

else if(source[i] ==')')

{

while((top !=-1) && (st[top] !='('))

{

target[j]=pop(st);

j++;

}

if(top==-1)

{

printf("\n Incorrect Expression");

exit(1);

}

temp=pop(st);

i++;

}

else if(isdigit(source[i]) || isalpha(source[i]))

{

target[j]=source[i];

j++;

i++;

}

else if(source[i] =='+' || source[i]=='-' || source[i]=='\*' || source[i]=='/'|| source[i]=='%')

{

while((top !=-1) && (st[top]!='(') && getpriority(st[top]) >= getpriority(source[i]))

{

target[j]=pop(st);

j++;

}

push(st,source[i]);

i++;

}

else

{

printf("\n Incorrect Expression");

exit(1);

}

}

while((top !=-1) && (st[top] !='('))

{

target[j]=pop(st);

j++;

}

target[j]='\0';

}

int getpriority(char op)

{

if(op=='/' || op=='\*' || op=='%')

return 1;

else if(op=='+' || op=='-')

return 0;

}

void push(char st[],char val)

{

if(top==MAX-1)

printf("\n Stack Overflow");

else

{

top++;

st[top]=val;

}

}

char pop(char st[])

{

char val=' ';

if(top==-1)

printf("\n Stak Underflow");

else

{

val=st[top];

top--;

}

   return val;

}

Program-08

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define MAX 5

int queue[MAX];

int front=-1,rear=-1;

void insert\_rear();

int delete\_front();

void display();

int peek();

void main()

{

int ch,item;

clrscr();

for(;;)

{

printf("Press 1 for Insert \n 2 for Delete \n 3 for display\n");

printf("Press 4 for Peek\n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: insert\_rear();

break;

case 2 : item=delete\_front();

if(item!=-1)

printf("Item deleted=%d",item);

break;

case 3 : display();

break;

case 4:item=peek();

if(item!=-1)

printf("First Value in the Queue=%d",item);

break;

default : exit(0);

}

}

getch();

}

void insert\_rear()

{

int num;

printf("\n Enter the item to be Inserted");

scanf("%d",&num);

if(rear==MAX-1)

printf("Queue is full\n");

else if(front==-1 && rear==-1)

front=rear=0;

else

rear++;

queue[rear]=num;

}

int delete\_front()

{

int item\_deleted;

if(front==-1 || front >rear)

{

printf("QUEUE is empty\n");

return -1;

}

else

{

item\_deleted=queue[front];

front++;

if(front>rear)

{

front=rear=-1;

}

return item\_deleted;

}

}

void display()

{

int i;

if(front==-1 || front >rear)

printf("QUEUE is empty\n");

else

{

printf("Content of the QUEUE is\n");

for(i=front;i<=rear;i++)

printf("%d\t",queue[i]);

}

}

int peek()

{

if(front==-1 || front >rear)

printf("QUEUE is empty\n");

else

{

return queue[front];

}

}

Program-09

#include < stdio.h >

#include < stdlib.h >

#include<string.h>

struct node {

int data;

int priority;

struct node \* next;

};

struct node \* start = NULL;

struct node \*insert(struct node \*start)

{

int val,pri;

struct node \*ptr,\*prev,\*cur;

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

printf("Enter the value and priority:\n");

scanf("%d%d",&val,&pri);

ptr->data=val;

ptr ->priority = pri;

if (start == NULL || pri < start->priority)

{

ptr->next=start;

start=ptr;

}

else

{

prev=NULL;

cur=start;

while(cur != NULL && pri >= cur->priority)

{

prev=cur;

cur=cur->next;

}

prev->next=ptr;

ptr->next=cur;

}

return start;

}

struct node \*dequeue(struct node \*start)

{

struct node \*ptr;

if (start == NULL)

{

printf("\nUnderflow\n");

exit(0);

}

else

{

ptr=start;

printf("\n Deleted item is:%d",ptr->data);

start=start->next;

free(ptr);

}

return start;

}

void display(struct node \*start)

{

struct node \*ptr;

ptr=start;

if (start == NULL)

{

printf("\nQueue is Empty\n");

}

else

{

printf("The queue is \n");

while (ptr!=NULL)

{

printf("\t%d[priority=%d]", ptr-> data,ptr->priority);

ptr = ptr -> next;

}

}

}

int main()

{

int choice;

printf("\nImplementation of Queue using Linked List\n");

while (choice != 4)

{

printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

printf("\nEnter your choice : ");

scanf("%d", & choice);

switch (choice)

{

case 1: start=insert(start);

break;

case 2:

start=dequeue(start);

break;

case 3:

display(start);

break;

case 4:

exit(0);

break;

default:

printf("\nWrong Choice\n");

}

}

return 0;

}

Program-10

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

#include<process.h>

#include<string.h>

struct node

{

int info;

struct node \*llink;

struct node \*rlink;

};

typedef struct node\* NODE;

NODE insert(int,NODE);

NODE getnode();

NODE minimum(NODE);

void inorder(NODE);

void preorder(NODE);

void postorder(NODE);

void iterative\_search(int,NODE);

NODE delete\_item(int,NODE);

void main()

{

int ch,item,flag,item\_deleted;

NODE root=NULL,min;

clrscr();

for(;;)

{

printf("Press 1 .INSERT \n 2. Inorder\n 3 preorder\n");

printf("4.postorder\n5:Search\n6:Find minimum\n7.delete\n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter the item to be inserted\n");

scanf("%d",&item);

root=insert(item,root);

break;

case 2: printf("Inorder tree traversal\n");

inorder(root);

break;

case 3: printf("preorder tree traversal\n");

preorder(root);

break;

case 4: printf("postorder tree traversal\n");

postorder(root);

break;

case 5: if(root==NULL)

printf("Tree is empty\n");

else

{

printf("Enter the item to be search\n");

scanf("%d",&item);

iterative\_search(item,root);

}

break;

case 6: min=minimum(root);

printf("smallest element =%d\n",min->info);

break;

case 7: printf("Enter the node to be deleted\n");

scanf("%d",&item\_deleted);

root=delete\_item(item,root);

break;

/\* case 3: display(head);

break; \*/

default: exit(0);

}//switch

}//for

getch();

}//main

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("Out of memeory\n");

exit(0);

}

return x;

}

void freenode(NODE temp)

{

free(temp);

}

NODE minimum(NODE root)

{

NODE cur;

if(root==NULL) return root;

cur=root;

while(cur->llink!=NULL)

cur=cur->llink;

return cur;

}

NODE insert(int item,NODE root)

{

NODE temp,cur,prev;

temp=getnode();

temp->info=item;

temp->llink=temp->rlink=NULL;

if(root==NULL) return temp;

prev=NULL;

cur=root;

while( cur !=NULL)

{

prev=cur;

if(item <cur->info)

cur=cur->llink;

else

cur=cur->rlink;

}

if(item< prev->info)

prev->llink=temp;

else

prev->rlink=temp;

return root;

}

void inorder(NODE root)

{

if(root !=NULL)

{

inorder(root->llink);

printf("%d ",root->info);

inorder(root->rlink);

}

}

void preorder(NODE root)

{

if(root !=NULL)

{

printf("%d ",root->info);

preorder(root->llink);

preorder(root->rlink);

}

}

void postorder(NODE root)

{

if(root !=NULL)

{

postorder(root->llink);

postorder(root->rlink);

printf("%d ",root->info);

}

}

void iterative\_search(int item,NODE root)

{

while(root !=NULL && item != root->info)

{

if(item< root->info)

root=root->llink;

else

root=root->rlink;

}

if(item==root->info)

printf("Succesfull search\n");

else

printf("Unsuccesful Search\n");

// return root;

}

NODE delete\_item(int item,NODE root)

{

NODE parent,cur,suc,psuc,ptr;

if(root->llink==NULL)

{

printf("Tree is empty\n");

return root;

}

parent=root;

cur=root->llink;

while(cur !=NULL && item !=cur->info)

{

parent=cur;

if(item<cur->info)

cur=cur->llink;

else

cur=cur->rlink;

}

if(cur==NULL)

{

printf("Item not found\n");

return root;

}

if(cur->llink ==NULL)

ptr=cur->rlink;

else if(cur->rlink ==NULL)

ptr=cur->llink;

else

{

psuc=cur;

cur=cur->llink;

while(suc->llink!=NULL)

{

psuc=cur;

suc=suc->llink;

}

if(cur==psuc)

{

suc->llink=cur->rlink;

}

else

{

suc->llink=cur->llink;

psuc->llink=suc->rlink;

suc->rlink=cur->rlink;

}

ptr=suc;

}

if(parent->llink==cur)

parent->llink=ptr;

else

parent->rlink=ptr;

freenode(cur);

return root;

}