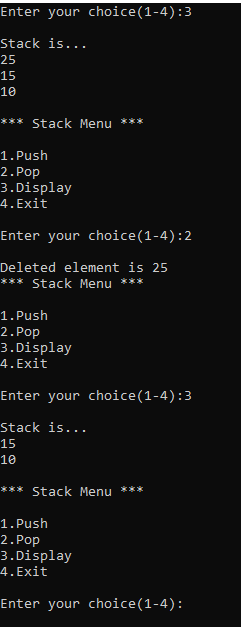
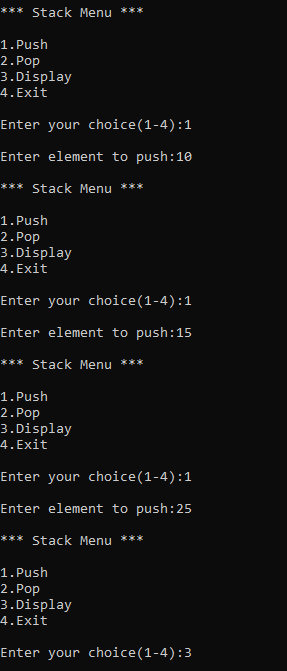
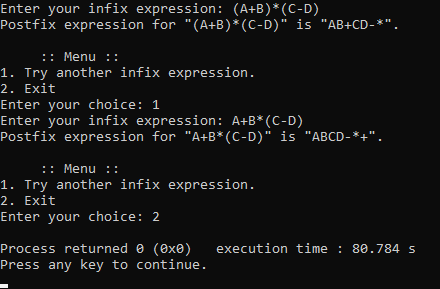
LAB1

|  |
| --- |
| #include<stdio.h> |
|  | #include<process.h> |
|  | #include<stdlib.h> |
|  |  |
|  | #define MAX 5 //Maximum number of elements that can be stored |
|  |  |
|  | int top=-1,stack[MAX]; |
|  | void push(); |
|  | void pop(); |
|  | void display(); |
|  |  |
|  | void main() |
|  | { |
|  | int ch; |
|  |  |
|  | while(1) //infinite loop, will end when choice will be 4 |
|  | { |
|  | printf("\n\*\*\* Stack Menu \*\*\*"); |
|  | printf("\n\n1.Push\n2.Pop\n3.Display\n4.Exit"); |
|  | printf("\n\nEnter your choice(1-4):"); |
|  | scanf("%d",&ch); |
|  |  |
|  | switch(ch) |
|  | { |
|  | case 1: push(); |
|  | break; |
|  | case 2: pop(); |
|  | break; |
|  | case 3: display(); |
|  | break; |
|  | case 4: exit(0); |
|  |  |
|  | default: printf("\nWrong Choice!!"); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void push() |
|  | { |
|  | int val; |
|  |  |
|  | if(top==MAX-1) |
|  | { |
|  | printf("\nStack is full!!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nEnter element to push:"); |
|  | scanf("%d",&val); |
|  | top=top+1; |
|  | stack[top]=val; |
|  | } |
|  | } |
|  |  |
|  | void pop() |
|  | { |
|  | if(top==-1) |
|  | { |
|  | printf("\nStack is empty!!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nDeleted element is %d",stack[top]); |
|  | top=top-1; |
|  | } |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | int i; |
|  |  |
|  | if(top==-1) |
|  | { |
|  | printf("\nStack is empty!!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nStack is...\n"); |
|  | for(i=top;i>=0;--i) |
|  | printf("%d\n",stack[i]); |
|  | } |
|  | } |



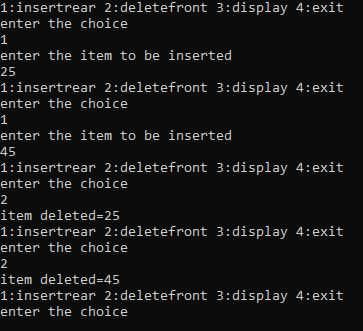
LAB2

|  |
| --- |
| #include <stdio.h> |
|  | #define N 100 |
|  |  |
|  | int stack[N]; |
|  |  |
|  | int top = -1; |
|  |  |
|  | void push(int item){ |
|  | if(top==N-1) |
|  | printf("Stack overflow!\n"); |
|  | else |
|  | stack[++top] = item; |
|  | } |
|  |  |
|  | int pop(){ |
|  | if(top==-1) |
|  | printf("Stack underflow!\n"); |
|  | else |
|  | return stack[top--]; |
|  |  |
|  | } |
|  |  |
|  | int priority(char op){ |
|  | switch(op){ |
|  | case '\*': return 2; |
|  | break; |
|  | case '/': return 2; |
|  | break; |
|  | case '+': return 1; |
|  | break; |
|  | case '-': return 1; |
|  | break; |
|  | case '(': return 0; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | char s[50]; |
|  | char t[50]; |
|  | int l; |
|  | int choice = 1; |
|  |  |
|  | do{ |
|  | l = 0; |
|  | printf("Enter your infix expression: "); |
|  | scanf("%s", s); |
|  |  |
|  | for(int i=0; s[i]!='\0'; i++){ |
|  | switch(s[i]){ |
|  | case '(': push('('); |
|  | break; |
|  | case ')': while(stack[top]!='('){ |
|  | t[l++] = pop(); |
|  | } |
|  | pop(); |
|  | break; |
|  | case '\*': |
|  | case '/': |
|  | case '+': |
|  | case '-': while(top!=-1 && priority(stack[top])>=priority(s[i])){ |
|  | t[l++] = pop(); |
|  | } |
|  | push(s[i]); |
|  | break; |
|  | default: t[l++] = s[i]; |
|  | } |
|  | } |
|  |  |
|  | while(top!=-1){ |
|  | t[l++] = pop(); |
|  | } |
|  |  |
|  | t[l] = '\0'; |
|  |  |
|  | printf("Postfix expression for \"%s\" is \"%s\".\n", s, t); |
|  |  |
|  | printf("\n :: Menu :: \n1. Try another infix expression.\n2. Exit\nEnter your choice: "); |
|  |  |
|  | scanf("%d", &choice); |
|  | }while(choice!=2); |
|  |  |
|  | return 0; |
|  | } |



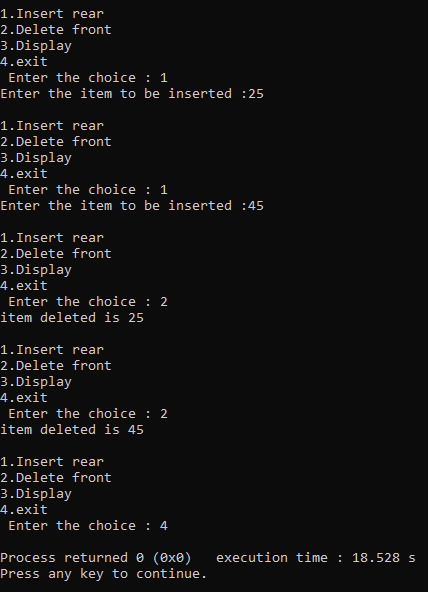
LAB 3

|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  | #define QUE\_SIZE 3 |
|  | int item,front=0,rear=-1,q[10]; |
|  | void insertrear() |
|  | {if(rear==QUE\_SIZE-1) |
|  | { |
|  | printf("queue overflow\n"); |
|  | return; |
|  | } |
|  | rear=rear+1; |
|  | q[rear]=item; |
|  | }int deletefront() |
|  | {if (front>rear) |
|  | {front=0; |
|  | rear=-1; |
|  | return -1; |
|  | }return q[front++]; |
|  | }void displayQ() |
|  | {int i; |
|  | if (front>rear) |
|  | { |
|  | printf("queue is empty\n"); |
|  | return; |
|  | } |
|  | printf("contents of queue\n"); |
|  | for(i=front;i<=rear;i++) |
|  | { |
|  | printf("%d\n",q[i]); |
|  | }} |
|  | int main() |
|  | { |
|  | int choice; |
|  | for(;;) |
|  | { |
|  | printf("1:insertrear 2:deletefront 3:display 4:exit\n"); |
|  | printf("enter the choice\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1:printf("enter the item to be inserted\n"); |
|  | scanf("%d",&item); |
|  | insertrear (); |
|  | break; |
|  | case 2:item=deletefront(); |
|  | if(item==-1) |
|  | printf("queue is empty\n"); |
|  | else |
|  | printf("item deleted=%d\n",item); |
|  | break; |
|  | case 3:displayQ(); |
|  | break; |
|  | default:exit (0); |
|  |  |
|  | } |
|  |  |
|  | } |
|  | } |



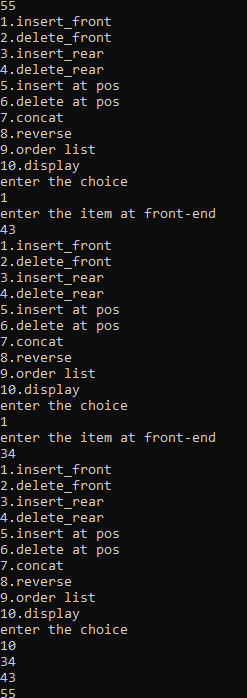
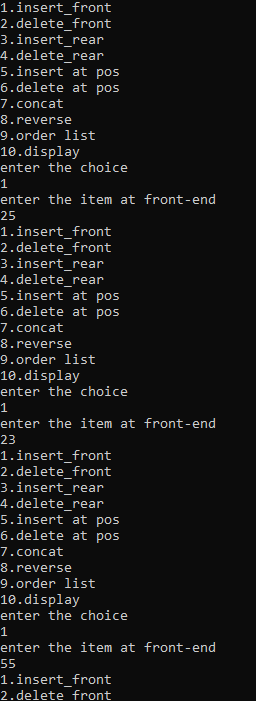
LAB 4

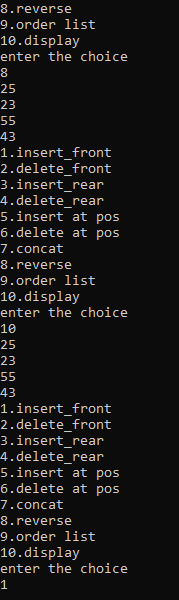
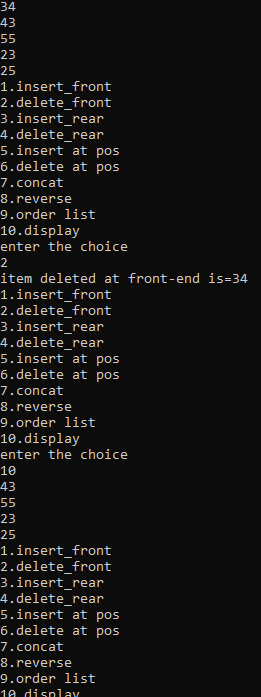
|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  | #include<process.h> |
|  | #define que\_size 3 |
|  | int item,front=0,rear=-1,q[que\_size],count=0; |
|  | void insertrear() |
|  | { |
|  | if(count==que\_size) |
|  | { |
|  | printf("queue overflow"); |
|  | return; |
|  | } |
|  | rear=(rear+1)%que\_size; |
|  | q[rear]=item; |
|  | count++; |
|  | } |
|  | int deletefront() |
|  | { |
|  | if(count==0) return -1; |
|  | item = q[front]; |
|  | front=(front+1)%que\_size; |
|  | count=count-1; |
|  | return item; |
|  | } |
|  | void displayq() |
|  | { |
|  | int i,f; |
|  | if(count==0) |
|  | { |
|  | printf("queue is empty"); |
|  | return; |
|  | } |
|  | f=front; |
|  | printf("contents of queue \n"); |
|  | for(i=0;i<=count;i++) |
|  | { |
|  | printf("%d\n",q[f]); |
|  | f=(f+1)%que\_size; |
|  | } |
|  | } |
|  | void main() |
|  | { |
|  | int choice; |
|  | for(;;) |
|  | { |
|  | printf("\n1.Insert rear \n2.Delete front \n3.Display \n4.exit \n "); |
|  | printf("Enter the choice : "); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1:printf("Enter the item to be inserted :"); |
|  | scanf("%d",&item); |
|  | insertrear(); |
|  | break; |
|  | case 2:item=deletefront(); |
|  | if(item==-1) |
|  | printf("queue is empty\n"); |
|  | Else |
|  | printf("item deleted is %d \n",item); |
|  | break; |
|  | case 3:displayq(); |
|  | break; |
|  | default:exit(0); |
|  | } |
|  | } |
|  | getch(); |
|  | } |



LAB 5 AND LAB 6

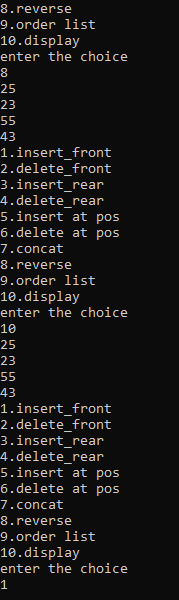
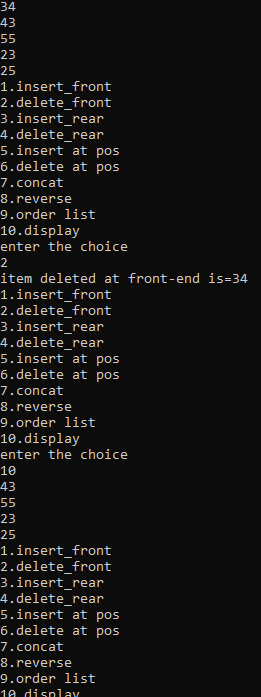
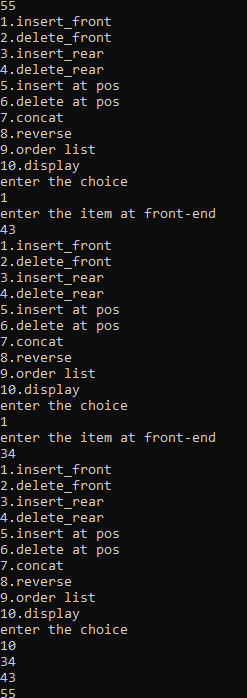
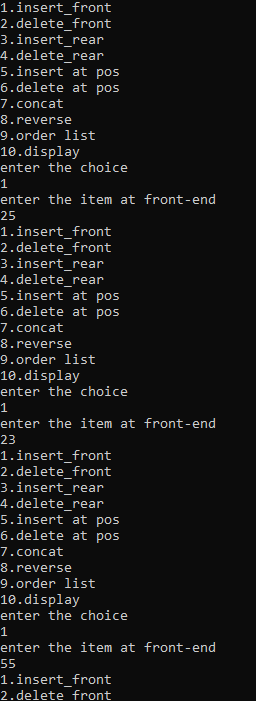
|  |
| --- |
| #include<stdio.h> |
|  | #include <stdlib.h> |
|  | struct node |
|  | { |
|  | int info; |
|  | struct node \*link; |
|  | }; |
|  | typedef struct node \*NODE; |
|  | NODE getnode() |
|  | { |
|  | NODE x; |
|  | x=(NODE)malloc(sizeof(struct node)); |
|  | if(x==NULL) |
|  | { |
|  | printf("mem full\n"); |
|  | exit(0); |
|  | } |
|  | return x; |
|  | } |
|  | void freenode(NODE x) |
|  | { |
|  | free(x); |
|  | } |
|  | NODE insert\_front(NODE first,int item) |
|  | { |
|  | NODE temp; |
|  | temp=getnode(); |
|  | temp->info=item; |
|  | temp->link=NULL; |
|  | if(first==NULL) |
|  | return temp; |
|  | temp->link=first; |
|  | first=temp; |
|  | return first; |
|  | } |
|  | NODE delete\_front(NODE first) |
|  | { |
|  | NODE temp; |
|  | if(first==NULL) |
|  | { |
|  | printf("list is empty cannot delete\n"); |
|  | return first; |
|  | } |
|  | temp=first; |
|  | temp=temp->link; |
|  | printf("item deleted at front-end is=%d\n",first->info); |
|  | free(first); |
|  | return temp; |
|  | } |
|  | NODE insert\_rear(NODE first,int item) |
|  | { |
|  | NODE temp,cur; |
|  | temp=getnode(); |
|  | temp->info=item; |
|  | temp->link=NULL; |
|  | if(first==NULL) |
|  | return temp; |
|  | cur=first; |
|  | while(cur->link!=NULL) |
|  | cur=cur->link; |
|  | cur->link=temp; |
|  | return first; |
|  | } |
|  | NODE delete\_rear(NODE first) |
|  | { |
|  | NODE cur,prev; |
|  | if(first==NULL) |
|  | { |
|  | printf("list is empty cannot delete\n"); |
|  | return first; |
|  | } |
|  | if(first->link==NULL) |
|  | { |
|  | printf("item deleted is %d\n",first->info); |
|  | free(first); |
|  | return NULL; |
|  | } |
|  | prev=NULL; |
|  | cur=first; |
|  | while(cur->link!=NULL) |
|  | { |
|  | prev=cur; |
|  | cur=cur->link; |
|  | } |
|  | printf("iten deleted at rear-end is %d",cur->info); |
|  | free(cur); |
|  | prev->link=NULL; |
|  | return first; |
|  | } |
|  | NODE insert\_pos(int item,int pos,NODE first) |
|  | { |
|  | NODE temp; |
|  | NODE prev,cur; |
|  | int count; |
|  | temp=getnode(); |
|  | temp->info=item; |
|  | temp->link=NULL; |
|  | if(first==NULL && pos==1) |
|  | return temp; |
|  | if(first==NULL) |
|  | { |
|  | printf("invalid pos\n"); |
|  | return first; |
|  | } |
|  | if(pos==1) |
|  | { |
|  | temp->link=first; |
|  | return temp; |
|  | } |
|  | count=1; |
|  | prev=NULL; |
|  | cur=first; |
|  | while(cur!=NULL && count!=pos) |
|  | { |
|  | prev=cur; |
|  | cur=cur->link; |
|  | count++; |
|  | } |
|  | if(count==pos) |
|  | { |
|  | prev->link=temp; |
|  | temp->link=cur; |
|  | return first; |
|  | } |
|  | printf("IP\n"); |
|  | return first; |
|  | } |
|  | NODE delete\_pos(int pos, NODE first){ |
|  | if (first == NULL){ |
|  | printf("List empty\n"); |
|  | return first; |
|  | } |
|  | NODE temp= first; |
|  | if (pos==1) |
|  | { |
|  | first = temp->link; |
|  | free(temp); |
|  | return first; |
|  | } |
|  | NODE prev; |
|  |  |
|  | for (int i=1; temp!=NULL && i<pos; i++){ |
|  | prev=temp; |
|  | temp = temp->link; |
|  | } |
|  | if (temp == NULL || temp->link == NULL){ |
|  | printf("Invalid position\n"); |
|  | return NULL; |
|  | } |
|  | prev->link=temp->link; |
|  | printf("Element deleted %d\n",temp->info); |
|  | free(temp); |
|  | return first; |
|  | } |
|  | void display(NODE first) |
|  | { |
|  | NODE temp; |
|  | if(first==NULL) |
|  | printf("list empty cannot display items\n"); |
|  | for(temp=first;temp!=NULL;temp=temp->link) |
|  | { |
|  | printf("%d\n",temp->info); |
|  | } |
|  | } |
|  | NODE concat(NODE first,NODE second) |
|  | { |
|  | NODE cur; |
|  | if(first==NULL) |
|  | return second; |
|  | if(second==NULL) |
|  | return first; |
|  | cur=first; |
|  | while(cur->link!=NULL) |
|  | cur=cur->link; |
|  | cur->link=second; |
|  | return first; |
|  | } |
|  | NODE reverse(NODE first) |
|  | { |
|  | NODE cur,temp; |
|  | cur=NULL; |
|  | while(first!=NULL) |
|  | { |
|  | temp=first; |
|  | first=first->link; |
|  | temp->link=cur; |
|  | cur=temp; |
|  | } |
|  | return cur; |
|  | } |
|  | NODE order\_list(NODE first) |
|  | { |
|  | int swapped, i; |
|  | NODE ptr1,lptr=NULL; |
|  | if (first == NULL) |
|  | return first; |
|  |  |
|  | Do |
|  | { |
|  | swapped = 0; |
|  | ptr1 = first; |
|  |  |
|  | while (ptr1->link != lptr) |
|  | { |
|  | if (ptr1->info > ptr1->link->info) |
|  | { |
|  | int temp = ptr1->info; |
|  | ptr1->info = ptr1->link->info; |
|  | ptr1->link->info = temp; |
|  | swapped = 1; |
|  | } |
|  | ptr1 = ptr1->link; |
|  | } |
|  | lptr = ptr1; |
|  | } |
|  | while (swapped); |
|  | return first; |
|  | } |
|  | void main() |
|  | { |
|  | int item,choice,pos,i,n; |
|  | NODE a,b; |
|  | NODE first=NULL; |
|  | for(;;) |
|  | { |
|  | printf("1.insert\_front\n2.delete\_front\n3.insert\_rear\n4.delete\_rear\n5.insert at pos\n6.delete at pos\n7.concat\n8.reverse\n9.order list\n10.display\n"); |
|  | printf("enter the choice\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1:printf("enter the item at front-end\n"); |
|  | scanf("%d",&item); |
|  | first=insert\_front(first,item); |
|  | break; |
|  | case 2:first=delete\_front(first); |
|  | break; |
|  | case 3:printf("enter the item at rear-end\n"); |
|  | scanf("%d",&item); |
|  | first=insert\_rear(first,item); |
|  | break; |
|  | case 4:first=delete\_rear(first); |
|  | break; |
|  | case 5: |
|  | printf("Enter item\n"); |
|  | scanf("%d",&item); |
|  | printf("enter the position\n"); |
|  | scanf("%d",&pos); |
|  | first=insert\_pos(item,pos,first); |
|  | break; |
|  | case 6: |
|  | printf("Enter posititon of deletion\n"); |
|  | scanf("%d",&pos); |
|  | first=delete\_pos(pos,first); |
|  | break; |
|  | case 7: |
|  | printf("enter the no of nodes in 1\n"); |
|  | scanf("%d",&n); |
|  | a=NULL; |
|  | for(i=0;i<n;i++) |
|  | { |
|  | printf("enter the item\n"); |
|  | scanf("%d",&item); |
|  | a=insert\_rear(a,item); |
|  | } |
|  | printf("enter the no of nodes in 2\n"); |
|  | scanf("%d",&n); |
|  | b=NULL; |
|  | for(i=0;i<n;i++) |
|  | { |
|  | printf("enter the item\n"); |
|  | scanf("%d",&item); |
|  | b=insert\_rear(b,item); |
|  | } |
|  | a=concat(a,b); |
|  | display(a); |
|  | break; |
|  | case 8: |
|  | first=reverse(first); |
|  | display(first); |
|  | break; |
|  | case 9: |
|  | first=order\_list(first); |
|  | break; |
|  | case 10:display(first); |
|  | break; |
|  | default:exit(0); |
|  | break; |
|  | } |
|  | } |
|  | } |





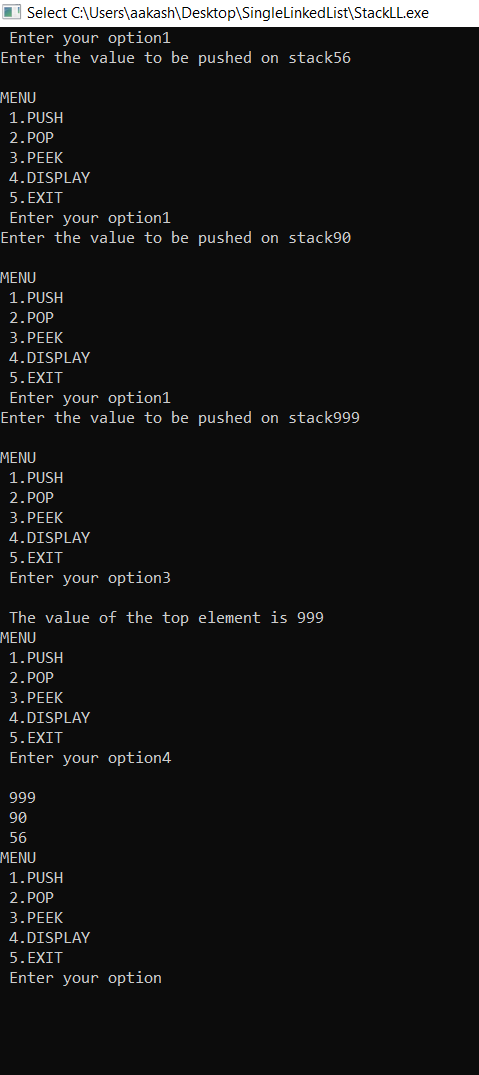
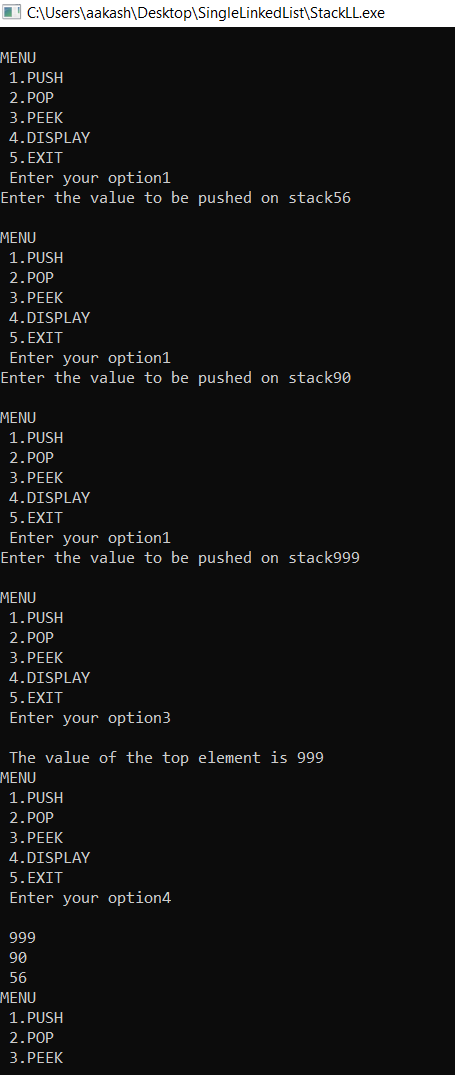
LAB 7

|  |
| --- |
| #include<stdio.h> |
|  | #include <stdlib.h> |
|  | struct node |
|  | { |
|  | int info; |
|  | struct node \*link; |
|  | }; |
|  | typedef struct node \*NODE; |
|  | NODE getnode() |
|  | { |
|  | NODE x; |
|  | x=(NODE)malloc(sizeof(struct node)); |
|  | if(x==NULL) |
|  | { |
|  | printf("mem full\n"); |
|  | exit(0); |
|  | } |
|  | return x; |
|  | } |
|  | void freenode(NODE x) |
|  | { |
|  | free(x); |
|  | } |
|  | NODE insert\_front(NODE first,int item) |
|  | { |
|  | NODE temp; |
|  | temp=getnode(); |
|  | temp->info=item; |
|  | temp->link=NULL; |
|  | if(first==NULL) |
|  | return temp; |
|  | temp->link=first; |
|  | first=temp; |
|  | return first; |
|  | } |
|  | NODE delete\_front(NODE first) |
|  | { |
|  | NODE temp; |
|  | if(first==NULL) |
|  | { |
|  | printf("list is empty cannot delete\n"); |
|  | return first; |
|  | } |
|  | temp=first; |
|  | temp=temp->link; |
|  | printf("item deleted at front-end is=%d\n",first->info); |
|  | free(first); |
|  | return temp; |
|  | } |
|  | NODE insert\_rear(NODE first,int item) |
|  | { |
|  | NODE temp,cur; |
|  | temp=getnode(); |
|  | temp->info=item; |
|  | temp->link=NULL; |
|  | if(first==NULL) |
|  | return temp; |
|  | cur=first; |
|  | while(cur->link!=NULL) |
|  | cur=cur->link; |
|  | cur->link=temp; |
|  | return first; |
|  | } |
|  | NODE delete\_rear(NODE first) |
|  | { |
|  | NODE cur,prev; |
|  | if(first==NULL) |
|  | { |
|  | printf("list is empty cannot delete\n"); |
|  | return first; |
|  | } |
|  | if(first->link==NULL) |
|  | { |
|  | printf("item deleted is %d\n",first->info); |
|  | free(first); |
|  | return NULL; |
|  | } |
|  | prev=NULL; |
|  | cur=first; |
|  | while(cur->link!=NULL) |
|  | { |
|  | prev=cur; |
|  | cur=cur->link; |
|  | } |
|  | printf("iten deleted at rear-end is %d",cur->info); |
|  | free(cur); |
|  | prev->link=NULL; |
|  | return first; |
|  | } |
|  | NODE insert\_pos(int item,int pos,NODE first) |
|  | { |
|  | NODE temp; |
|  | NODE prev,cur; |
|  | int count; |
|  | temp=getnode(); |
|  | temp->info=item; |
|  | temp->link=NULL; |
|  | if(first==NULL && pos==1) |
|  | return temp; |
|  | if(first==NULL) |
|  | { |
|  | printf("invalid pos\n"); |
|  | return first; |
|  | } |
|  | if(pos==1) |
|  | { |
|  | temp->link=first; |
|  | return temp; |
|  | } |
|  | count=1; |
|  | prev=NULL; |
|  | cur=first; |
|  | while(cur!=NULL && count!=pos) |
|  | { |
|  | prev=cur; |
|  | cur=cur->link; |
|  | count++; |
|  | } |
|  | if(count==pos) |
|  | { |
|  | prev->link=temp; |
|  | temp->link=cur; |
|  | return first; |
|  | } |
|  | printf("IP\n"); |
|  | return first; |
|  | } |
|  | NODE delete\_pos(int pos, NODE first){ |
|  | if (first == NULL){ |
|  | printf("List empty\n"); |
|  | return first; |
|  | } |
|  | NODE temp= first; |
|  | if (pos==1) |
|  | { |
|  | first = temp->link; |
|  | free(temp); |
|  | return first; |
|  | } |
|  | NODE prev; |
|  |  |
|  | for (int i=1; temp!=NULL && i<pos; i++){ |
|  | prev=temp; |
|  | temp = temp->link; |
|  | } |
|  | if (temp == NULL || temp->link == NULL){ |
|  | printf("Invalid position\n"); |
|  | return NULL; |
|  | } |
|  | prev->link=temp->link; |
|  | printf("Element deleted %d\n",temp->info); |
|  | free(temp); |
|  | return first; |
|  | } |
|  | void display(NODE first) |
|  | { |
|  | NODE temp; |
|  | if(first==NULL) |
|  | printf("list empty cannot display items\n"); |
|  | for(temp=first;temp!=NULL;temp=temp->link) |
|  | { |
|  | printf("%d\n",temp->info); |
|  | } |
|  | } |
|  | NODE concat(NODE first,NODE second) |
|  | { |
|  | NODE cur; |
|  | if(first==NULL) |
|  | return second; |
|  | if(second==NULL) |
|  | return first; |
|  | cur=first; |
|  | while(cur->link!=NULL) |
|  | cur=cur->link; |
|  | cur->link=second; |
|  | return first; |
|  | } |
|  | NODE reverse(NODE first) |
|  | { |
|  | NODE cur,temp; |
|  | cur=NULL; |
|  | while(first!=NULL) |
|  | { |
|  | temp=first; |
|  | first=first->link; |
|  | temp->link=cur; |
|  | cur=temp; |
|  | } |
|  | return cur; |
|  | } |
|  | NODE order\_list(NODE first) |
|  | { |
|  | int swapped, i; |
|  | NODE ptr1,lptr=NULL; |
|  | if (first == NULL) |
|  | return first; |
|  |  |
|  | Do |
|  | { |
|  | swapped = 0; |
|  | ptr1 = first; |
|  |  |
|  | while (ptr1->link != lptr) |
|  | { |
|  | if (ptr1->info > ptr1->link->info) |
|  | { |
|  | int temp = ptr1->info; |
|  | ptr1->info = ptr1->link->info; |
|  | ptr1->link->info = temp; |
|  | swapped = 1; |
|  | } |
|  | ptr1 = ptr1->link; |
|  | } |
|  | lptr = ptr1; |
|  | } |
|  | while (swapped); |
|  | return first; |
|  | } |
|  | void main() |
|  | { |
|  | int item,choice,pos,i,n; |
|  | NODE a,b; |
|  | NODE first=NULL; |
|  | for(;;) |
|  | { |
|  | printf("1.insert\_front\n2.delete\_front\n3.insert\_rear\n4.delete\_rear\n5.insert at pos\n6.delete at pos\n7.concat\n8.reverse\n9.order list\n10.display\n"); |
|  | printf("enter the choice\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1:printf("enter the item at front-end\n"); |
|  | scanf("%d",&item); |
|  | first=insert\_front(first,item); |
|  | break; |
|  | case 2:first=delete\_front(first); |
|  | break; |
|  | case 3:printf("enter the item at rear-end\n"); |
|  | scanf("%d",&item); |
|  | first=insert\_rear(first,item); |
|  | break; |
|  | case 4:first=delete\_rear(first); |
|  | break; |
|  | case 5: |
|  | printf("Enter item\n"); |
|  | scanf("%d",&item); |
|  | printf("enter the position\n"); |
|  | scanf("%d",&pos); |
|  | first=insert\_pos(item,pos,first); |
|  | break; |
|  | case 6: |
|  | printf("Enter posititon of deletion\n"); |
|  | scanf("%d",&pos); |
|  | first=delete\_pos(pos,first); |
|  | break; |
|  | case 7: |
|  | printf("enter the no of nodes in 1\n"); |
|  | scanf("%d",&n); |
|  | a=NULL; |
|  | for(i=0;i<n;i++) |
|  | { |
|  | printf("enter the item\n"); |
|  | scanf("%d",&item); |
|  | a=insert\_rear(a,item); |
|  | } |
|  | printf("enter the no of nodes in 2\n"); |
|  | scanf("%d",&n); |
|  | b=NULL; |
|  | for(i=0;i<n;i++) |
|  | { |
|  | printf("enter the item\n"); |
|  | scanf("%d",&item); |
|  | b=insert\_rear(b,item); |
|  | } |
|  | a=concat(a,b); |
|  | display(a); |
|  | break; |
|  | case 8: |
|  | first=reverse(first); |
|  | display(first); |
|  | break; |
|  | case 9: |
|  | first=order\_list(first); |
|  | break; |
|  | case 10:display(first); |
|  | break; |
|  | default:exit(0); |
|  | break; |

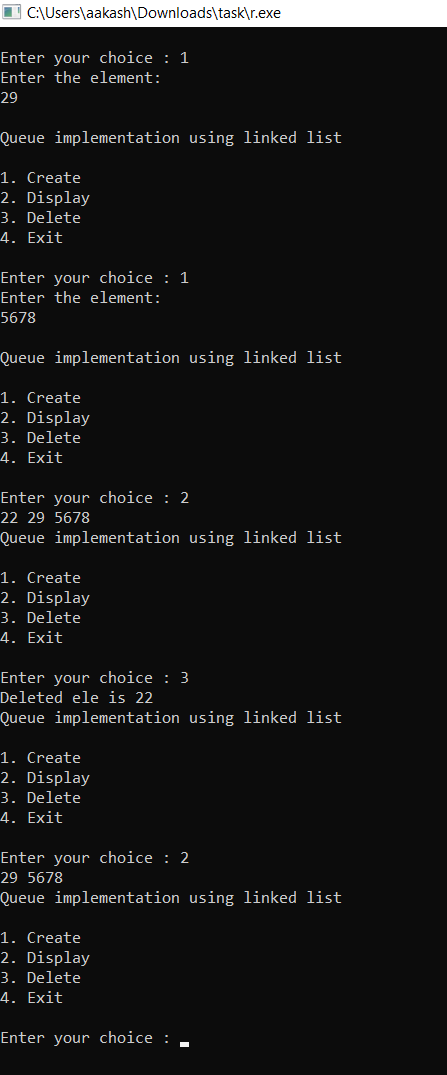


LAB 8

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <malloc.h> |
|  |  |
|  | struct stack{ |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  |  |
|  | struct stack \*top=NULL; |
|  |  |
|  | struct stack \*push(struct stack \*,int ); |
|  | struct stack \*display(struct stack \* ); |
|  | struct stack \*pop(struct stack \* ); |
|  | int peek(struct stack \*); |
|  |  |
|  | int main(){ |
|  | int val,option; |
|  | do{printf("\nMENU"); |
|  | printf("\n 1.PUSH"); |
|  | printf("\n 2.POP"); |
|  | printf("\n 3.PEEK"); |
|  | printf("\n 4.DISPLAY"); |
|  | printf("\n 5.EXIT"); |
|  | printf("\n Enter your option"); |
|  | scanf("%d",& option); |
|  | switch(option){ |
|  | case 1: |
|  | printf("Enter the value to be pushed on stack"); |
|  | scanf("%d",&val); |
|  | top=push(top,val); |
|  | break; |
|  | case 2: |
|  | top=pop(top); |
|  | break; |
|  | case 3: |
|  | val=peek(top); |
|  | if(val!=-1) |
|  | printf("\n The value of the top element is %d ",val); |
|  | else |
|  | printf("\n Stack is EMPTY"); |
|  | break; |
|  | case 4: |
|  | top=display(top); |
|  | break; |
|  | } |
|  | }while(option!=5); |
|  | return 0; |
|  | } |
|  |  |
|  | struct stack \*push (struct stack \*top,int val){ |
|  | struct stack \*p; |
|  | p=(struct stack \*)malloc (sizeof(struct stack)); |
|  | p->data=val; |
|  | if(top==0){ |
|  | p->next=0; |
|  | top=p; |
|  | } |
|  | else { |
|  | p->next=top; |
|  | top=p; |
|  | } |
|  | return top; |
|  | } |
|  |  |
|  | struct stack \*display (struct stack \*top){ |
|  | struct stack \*p; |
|  | p=top; |
|  | if(top==NULL) |
|  | printf("\n Stack is Empty"); |
|  | else { |
|  | while(p!=NULL) |
|  | { |
|  | printf("\n %d",p->data); |
|  | p=p->next; |
|  | } |
|  | } |
|  | return top; |
|  | } |
|  |  |
|  | struct stack \*pop(struct stack \*top){ |
|  | struct stack \*p; |
|  | p=top; |
|  | if(top==0) |
|  | printf("Stack Underflow"); |
|  | else { |
|  | top=top->next; |
|  | printf("\n The value of the item deleted is %d",p->data); |
|  | free(p); |
|  | } |
|  | return top; |
|  | } |
|  |  |
|  | int peek (struct stack \*top ){ |
|  | if(top==NULL) |
|  | return -1; |
|  | else |
|  | return top->data; |
|  | } |

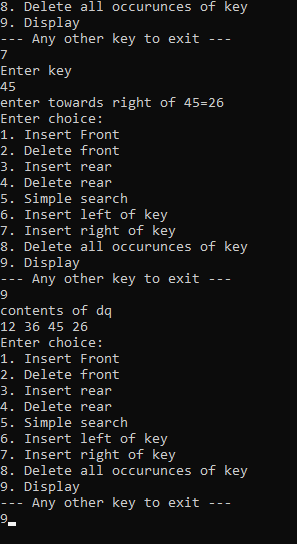
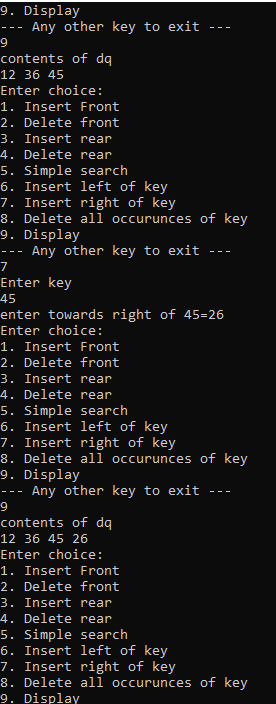
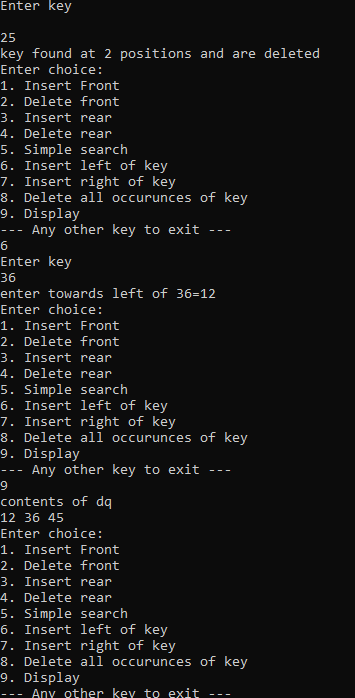
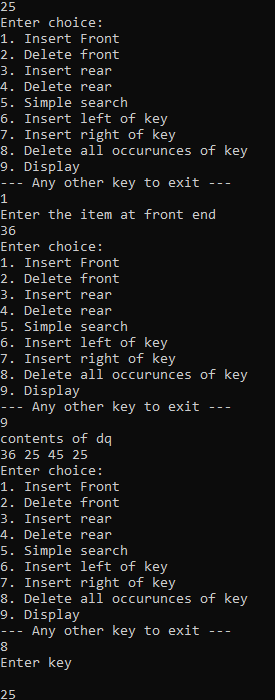
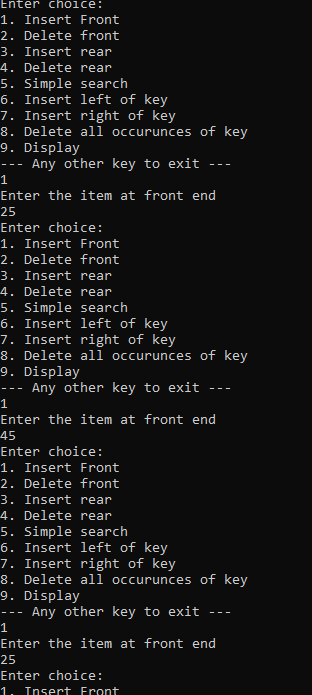


|  |
| --- |
| include <stdio.h> |
|  | #include <stdlib.h> |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  | void insert(); |
|  | void display(); |
|  | void del(); |
|  |  |
|  | struct node \*rear=NULL, \*front =NULL; |
|  |  |
|  | int main(int argc, char \*\*argv) |
|  | { |
|  | ; |
|  | int choice; |
|  | char ch = 'Y'; |
|  | do |
|  | { |
|  | printf("\nQueue implementation using linked list\n"); |
|  | printf("\n1. Create \n2. Display \n3. Delete \n4. Exit \n"); |
|  | printf("\nEnter your choice : "); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: insert(); break; |
|  | case 2: display();break; |
|  | case 3: del(); break; |
|  | case 4: |
|  | ch = 'n'; |
|  | break; |
|  | } |
|  | }while(ch=='y'||ch=='Y'); |
|  | } |
|  |  |
|  | void insert() |
|  | { |
|  | struct node \*newnode; |
|  | newnode=(struct node \*) malloc(sizeof(struct node)); |
|  | printf("Enter the element:\n"); |
|  | scanf("%d",&newnode->data); |
|  | newnode->next=NULL; |
|  |  |
|  | if(rear==NULL) |
|  | { |
|  | rear=newnode; |
|  | front=newnode; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | rear->next=newnode; |
|  | rear=newnode; |
|  | } |
|  | } |
|  |  |
|  | void del() |
|  | { |
|  | if(front==NULL) |
|  | { |
|  | printf("Queue is empty\n");return; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | printf("Deleted ele is %d",front->data); |
|  | if(front==rear) |
|  | { |
|  | printf("Queue is empty\n"); |
|  | front=NULL; rear=NULL; |
|  | } |
|  | else |
|  | front=front->next; |
|  | } |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | struct node \*temp; |
|  | if(front ==NULL) |
|  | { |
|  | printf("Queue is empty"); |
|  | return; |
|  | } |
|  | temp=front; |
|  | while (temp !=NULL) |
|  | { |
|  | printf("%d ",temp->data); |
|  | temp=temp->next; |
|  | } |
|  |  |
|  | } |



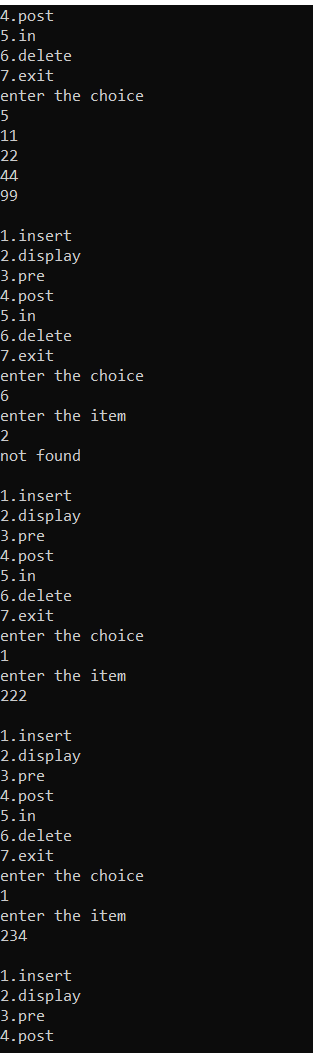
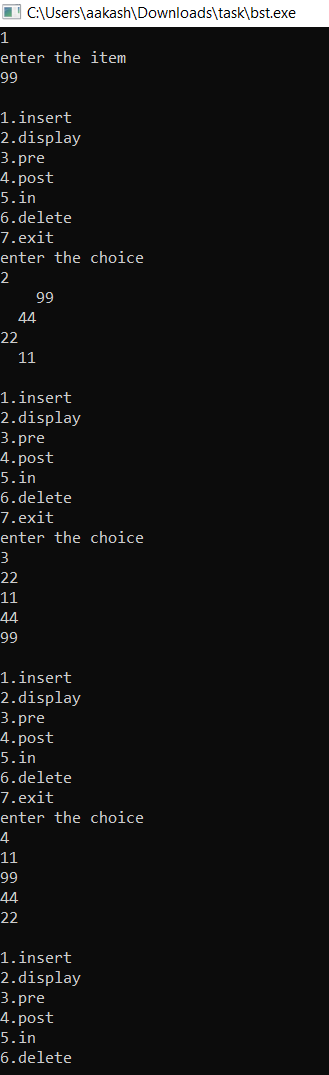
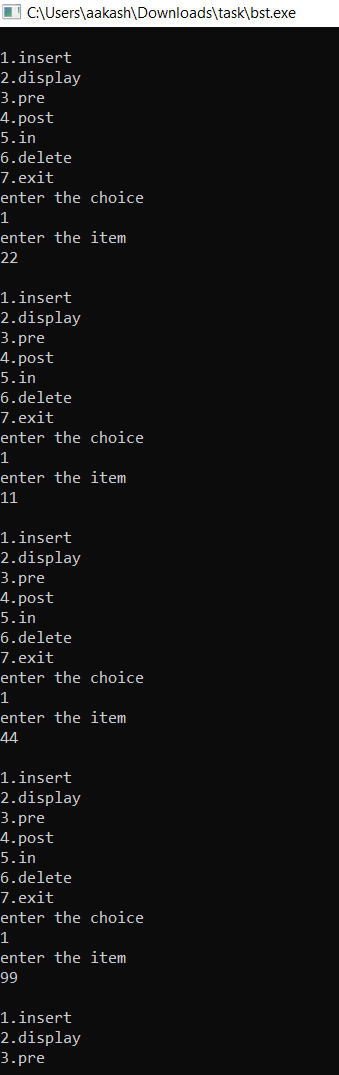
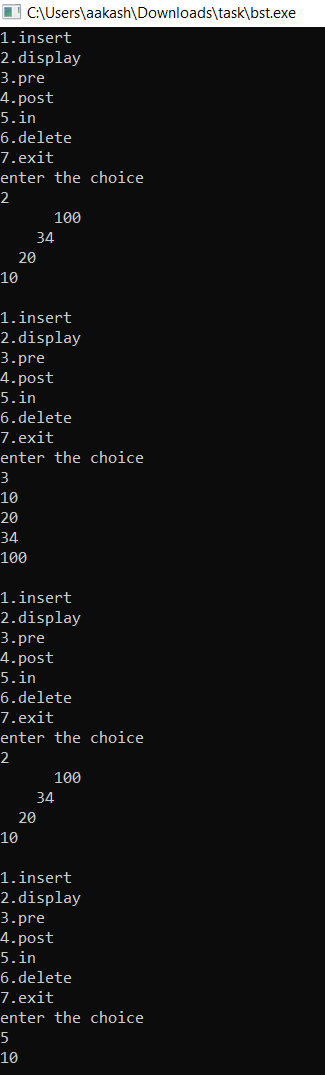
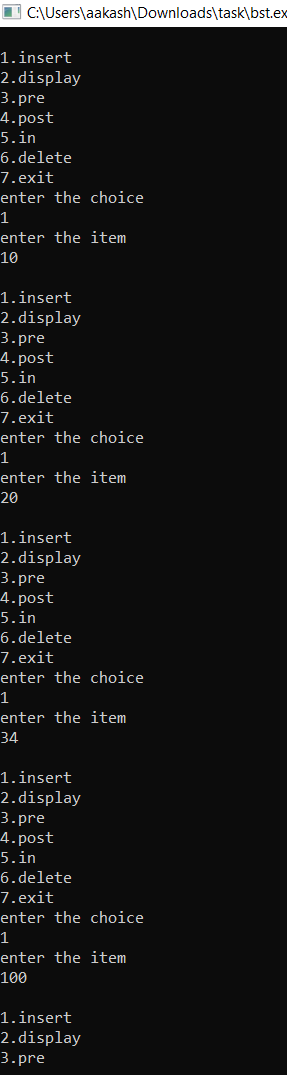
LAB 9

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | typedef struct node{ |
|  | struct node \*prev; |
|  | int data; |
|  | struct node \*next; |
|  | }\*NODE; |
|  |  |
|  | NODE makeNode(int x){ |
|  | NODE temp = (NODE)malloc(sizeof(struct node)); |
|  | temp->prev = NULL; |
|  | temp->data = x; |
|  | temp->next = NULL; |
|  | return temp; |
|  | } |
|  |  |
|  | /\* |
|  | Insert Functions |
|  | \*/ |
|  |  |
|  | NODE insertFront(NODE head){ |
|  | int ele; |
|  | printf("\nElement:"); |
|  | scanf("%d", &ele); |
|  |  |
|  | NODE temp = makeNode(ele); |
|  | temp->next = head; |
|  |  |
|  | return temp; |
|  | } |
|  |  |
|  | NODE insertRear(NODE head){ |
|  | int ele; |
|  | printf("\nElement:"); |
|  | scanf("%d", &ele); |
|  |  |
|  | NODE temp = makeNode(ele); |
|  |  |
|  | if(head == NULL){ |
|  | return temp; |
|  | } |
|  | else{ |
|  | NODE p = head; |
|  |  |
|  | // p will point to last element |
|  | while((p->next) != NULL){ |
|  | p = p->next; |
|  | } |
|  | p->next = temp; |
|  | temp->prev = p; |
|  |  |
|  | return head; |
|  | } |
|  | } |
|  |  |
|  | NODE insertPos(NODE head){ |
|  | int ele, pos; |
|  | printf("\nElement & Position: "); |
|  | scanf("%d %d", &ele, &pos); |
|  |  |
|  | NODE temp = makeNode(ele); |
|  |  |
|  | if(head == NULL){ |
|  | if(pos != 1) |
|  | printf("\nPosition doesnt exist!"); |
|  | else |
|  | return temp; |
|  | } |
|  | else{ |
|  | NODE p = head; |
|  |  |
|  | // p will point to pos-1'th element |
|  | for(int c=0; c<pos-1; c++){ |
|  | p = p->next; |
|  | } |
|  | // backup p->next |
|  | (p->next)->prev = temp; |
|  | temp->next = p->next; |
|  | p->next = temp; |
|  | temp->prev = p; |
|  |  |
|  | return head; |
|  | } |
|  | } |
|  |  |
|  | /\* |
|  | Delete Functions |
|  | \*/ |
|  |  |
|  | NODE deleteFront(NODE head){ |
|  | if(head == NULL) |
|  | printf("\nList Empty!"); |
|  | else{ |
|  | NODE temp = head->next; |
|  | if(temp == NULL){ |
|  | return NULL; |
|  | } |
|  | else{ |
|  | temp->prev = NULL; |
|  | free(head); |
|  | return temp; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | NODE deleteRear(NODE head){ |
|  | if(head == NULL) |
|  | printf("\nList Empty!"); |
|  | else{ |
|  | NODE temp = head; |
|  | // temp goto last but 1 ele |
|  | while((temp->next)->next != NULL){ |
|  | temp = temp->next; |
|  | } |
|  | free(temp->next); |
|  | temp->next == NULL; |
|  | return head; |
|  | } |
|  | } |
|  |  |
|  | NODE deletePos(NODE head){ |
|  | if(head == NULL) |
|  | printf("\nList Empty!"); |
|  | else{ |
|  | int pos; |
|  | NODE temp = head; |
|  | printf("Enter Position:"); |
|  | scanf("%d", &pos); |
|  |  |
|  | if(pos == 1) |
|  | head = deleteFront(head); |
|  | else{ |
|  | int i=1; |
|  | while(i < pos){ |
|  | if(temp->next != NULL) |
|  | temp = temp->next; |
|  | else{ |
|  | printf("Position doesnt exist!"); |
|  | return head; |
|  | } |
|  | i++; |
|  | } |
|  | NODE posnd = temp->next; |
|  | (posnd->next)->prev = temp; |
|  | temp->next = posnd->next; |
|  | free(posnd); |
|  | return head; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void display(NODE head){ |
|  | if(head == NULL) |
|  | printf("\nEmpty List!"); |
|  | else{ |
|  | NODE p = head; |
|  |  |
|  | printf("\nLIST >> "); |
|  | while(p != NULL){ |
|  | /\* data view \*/ |
|  | printf("%d ", p->data); |
|  |  |
|  | /\* full view \*/ |
|  | // printf("\n%d\t%d\t%d", &(p->prev), p->data, &(p->next)); |
|  |  |
|  | p = p->next; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void main(){ |
|  | NODE head = NULL; |
|  | int ch; |
|  |  |
|  | printf("\n\n---MENU---\n1.Ins Fr\n2.Ins Rr\n3.Ins Ps"); |
|  | printf("\n4.Del Fr\n5.Del Rr\n6.Del Ps\n8.Exit\n"); |
|  | while(1){ |
|  | // printf("\n\n---MENU---\n1.Ins Fr\n2.Ins Rr\n3.Ins Ps"); |
|  | // printf("\n4.Del Fr\n5.Del Rr\n6.Del Ps\n420.Exit\n"); |
|  | printf("\nChoice:"); |
|  | scanf("%d", &ch); |
|  | switch(ch){ |
|  | case 1: |
|  | head = insertFront(head); |
|  | display(head); |
|  | break; |
|  | case 2: |
|  | head = insertRear(head); |
|  | display(head); |
|  | break; |
|  | case 3: |
|  | head = insertPos(head); |
|  | display(head); |
|  | break; |
|  | case 4: |
|  | head = deleteFront(head); |
|  | display(head); |
|  | break; |
|  | case 5: |
|  | head = deleteRear(head); |
|  | display(head); |
|  | break; |
|  | case 6: |
|  | head = deletePos(head); |
|  | display(head); |
|  | break; |
|  | case 8: |
|  | printf("\nExiting!"); |
|  | exit(1); |
|  | default: |
|  | printf("\nWrong Input!"); |
|  | } |
|  | } |
|  | } |



LAB 10

|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  |  |
|  | struct node |
|  | { |
|  | int info; |
|  | struct node \*rlink; |
|  | struct node \*llink; |
|  | }; |
|  | typedef struct node \*NODE; |
|  | NODE getnode() |
|  | { |
|  | NODE x; |
|  | x=(NODE)malloc(sizeof(struct node)); |
|  | if(x==NULL) |
|  | { |
|  | printf("mem full\n"); |
|  | exit(0); |
|  | } |
|  | return x; |
|  | } |
|  | void freenode(NODE x) |
|  | { |
|  | free(x); |
|  | } |
|  | NODE insert(NODE root,int item) |
|  | { |
|  | NODE temp,cur,prev; |
|  | temp=getnode(); |
|  | temp->rlink=NULL; |
|  | temp->llink=NULL; |
|  | temp->info=item; |
|  | if(root==NULL) |
|  | return temp; |
|  | prev=NULL; |
|  | cur=root; |
|  | while(cur!=NULL) |
|  | { |
|  | prev=cur; |
|  | cur=(item<cur->info)?cur->llink:cur->rlink; |
|  | } |
|  | if(item<prev->info) |
|  | prev->llink=temp; |
|  | else |
|  | prev->rlink=temp; |
|  | return root; |
|  | } |
|  | void display(NODE root,int i) |
|  | { |
|  | int j; |
|  | if(root!=NULL) |
|  | { |
|  | display(root->rlink,i+1); |
|  | for(j=0;j<i;j++) |
|  | printf(" "); |
|  | printf("%d\n",root->info); |
|  | display(root->llink,i+1); |
|  | } |
|  | } |
|  | NODE delete(NODE root,int item) |
|  | { |
|  | NODE cur,parent,q,suc; |
|  | if(root==NULL) |
|  | { |
|  | printf("empty\n"); |
|  | return root; |
|  | } |
|  | parent=NULL; |
|  | cur=root; |
|  | while(cur!=NULL&&item!=cur->info) |
|  | { |
|  | parent=cur; |
|  | cur=(item<cur->info)?cur->llink:cur->rlink; |
|  | } |
|  | if(cur==NULL) |
|  | { |
|  | printf("not found\n"); |
|  | return root; |
|  | } |
|  | if(cur->llink==NULL) |
|  | q=cur->rlink; |
|  | else if(cur->rlink==NULL) |
|  | q=cur->llink; |
|  | else |
|  | { |
|  | suc=cur->rlink; |
|  | while(suc->llink!=NULL) |
|  | suc=suc->llink; |
|  | suc->llink=cur->llink; |
|  | q=cur->rlink; |
|  | } |
|  | if(parent==NULL) |
|  | return q; |
|  | if(cur==parent->llink) |
|  | parent->llink=q; |
|  | else |
|  | parent->rlink=q; |
|  | freenode(cur); |
|  | return root; |
|  | } |
|  |  |
|  | void preorder(NODE root) |
|  | { |
|  | if(root!=NULL) |
|  | { |
|  | printf("%d\n",root->info); |
|  | preorder(root->llink); |
|  | preorder(root->rlink); |
|  | } |
|  | } |
|  | void postorder(NODE root) |
|  | { |
|  | if(root!=NULL) |
|  | { |
|  |  |
|  | postorder(root->llink); |
|  | postorder(root->rlink); |
|  | printf("%d\n",root->info); |
|  | } |
|  | } |
|  | void inorder(NODE root) |
|  | { |
|  | if(root!=NULL) |
|  | { |
|  |  |
|  | inorder(root->llink); |
|  | printf("%d\n",root->info); |
|  | inorder(root->rlink); |
|  | } |
|  | } |
|  | void main() |
|  | { |
|  | int item,choice; |
|  | NODE root=NULL; |
|  | for(;;) |
|  | { |
|  | printf("\n1.insert\n2.display\n3.pre\n4.post\n5.in\n6.delete\n7.exit\n"); |
|  | printf("enter the choice\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1:printf("enter the item\n"); |
|  | scanf("%d",&item); |
|  | root=insert(root,item); |
|  | break; |
|  | case 2:display(root,0); |
|  | break; |
|  | case 3:preorder(root); |
|  | break; |
|  | case 4:postorder(root); |
|  | break; |
|  | case 5:inorder(root); |
|  | break; |
|  | case 6:printf("enter the item\n"); |
|  | scanf("%d",&item); |
|  | root=delete(root,item); |
|  | break; |
|  | default:exit(0); |
|  | break; |
|  | } |
|  | } |
|  | } |



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_XXXX\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_