**ADVANCED DATASTRUCTURE LAB CYCLES**

1. Program to implement doubly linked list.

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | #include<stdlib.h> |
|  | struct node |
|  | { |
|  | struct node \*prev; |
|  | struct node \*next; |
|  | int data; |
|  | }; |
|  | struct node \*head; |
|  | void insert\_begin(); |
|  | void insert\_last(); |
|  | void insert\_spec(); |
|  | void delete\_begin(); |
|  | void delete\_last(); |
|  | void delete\_spec(); |
|  | void show(); |
|  | void search(); |
|  | void main() |
|  | { |
|  | int ch; |
|  | while(ch!=9) |
|  | { |
|  | printf("\n1.insert in beginning\n2.insert at last\n3.insert randomly\n4.delete from beginning\n5.delete from last\n6.delete node after given node\n7.search\n8.show\n9.exit\n"); |
|  | printf("enter your choice :"); |
|  | scanf("%d",&ch); |
|  | switch(ch) |
|  | { |
|  | case 1: |
|  | insert\_begin(); |
|  | break; |
|  | case 2 : |
|  | insert\_last(); |
|  | break; |
|  | case 3: |
|  | insert\_spec(); |
|  | break; |
|  | case 4: |
|  | delete\_begin(); |
|  | break; |
|  | case 5: |
|  | delete\_last(); |
|  | break; |
|  | case 6: |
|  | delete\_spec(); |
|  | break; |
|  | case 7: |
|  | search(); |
|  | break; |
|  | case 8: |
|  | show(); |
|  | break; |
|  | case 9: |
|  | exit(0); |
|  | break; |
|  | default: |
|  | printf("invalid choice!"); |
|  | } |
|  | } |
|  | } |
|  | void insert\_begin() |
|  | { |
|  | struct node \*ptr; |
|  | int item; |
|  | ptr=(struct node\*)malloc(sizeof(struct node)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter value :"); |
|  | scanf("%d",&item); |
|  | if(head==NULL) |
|  | { |
|  | ptr->next=NULL; |
|  | ptr->prev=NULL; |
|  | ptr->data=item; |
|  | head=ptr; |
|  | } |
|  | else |
|  | { |
|  | ptr->data=item; |
|  | ptr->prev=NULL; |
|  | ptr->next=head; |
|  | head->prev=ptr; |
|  | head=ptr; |
|  | } |
|  | printf("\nnode inserted!"); |
|  | } |
|  | } |
|  | void insert\_last() |
|  | { |
|  | struct node \*ptr,\*temp; |
|  | int item; |
|  | ptr=(struct node\*)malloc(sizeof(struct node)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter value :"); |
|  | scanf("%d",&item); |
|  | ptr->data=item; |
|  | if(head==NULL) |
|  | { |
|  | ptr->next=NULL; |
|  | ptr->prev=NULL; |
|  | head=ptr; |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | temp->next=ptr; |
|  | ptr->prev=temp; |
|  | ptr->next=NULL; |
|  | } |
|  | } |
|  | printf("\nnode inserted!"); |
|  | } |
|  | void insert\_spec() |
|  | { |
|  | struct node \*ptr,\*temp; |
|  | int item,loc,i; |
|  | ptr=(struct node\*)malloc(sizeof(struct node)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow!"); |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | printf("\nenter the location :"); |
|  | scanf("%d",&loc); |
|  | for(i=0;i<loc;i++) |
|  | { |
|  | temp=temp->next; |
|  | if(temp==NULL) |
|  | { |
|  | printf("\nthere are less than %d items.",loc); |
|  | return; |
|  | } |
|  | printf("\nenter value :"); |
|  | scanf("%d",&item); |
|  | ptr->data=item; |
|  | ptr->next=temp->next; |
|  | ptr->prev=temp; |
|  | temp->next=ptr; |
|  | temp->next->prev=ptr; |
|  | printf("\nnode inserted"); |
|  | } |
|  | } |
|  | } |
|  | void delete\_begin() |
|  | { |
|  | struct node \*ptr; |
|  | if(head==NULL) |
|  | { |
|  | printf("\nunderflow!"); |
|  | } |
|  | else if(head->next==NULL) |
|  | { |
|  | head=NULL; |
|  | free(head); |
|  | printf("\nnode deleted!"); |
|  | } |
|  | else |
|  | { |
|  | ptr=head; |
|  | head=head->next; |
|  | head->prev=NULL; |
|  | free(ptr); |
|  | printf("\nnode deleted!"); |
|  | } |
|  | } |
|  | void delete\_last() |
|  | { |
|  | struct node \*ptr; |
|  | if(head==NULL) |
|  | { |
|  | printf("\nunderflow!"); |
|  | } |
|  | else if(head->next==NULL) |
|  | { |
|  | head=NULL; |
|  | free(head); |
|  | printf("\nnode deleted!"); |
|  | } |
|  | else |
|  | { |
|  | ptr=head; |
|  | if(ptr->next!=NULL) |
|  | { |
|  | ptr=ptr->next; |
|  | } |
|  | ptr->prev->next=NULL; |
|  | free(ptr); |
|  | printf("\nnode deleted!"); |
|  | } |
|  | } |
|  | void delete\_spec() |
|  | { |
|  | struct node \*ptr,\*temp; |
|  | int val; |
|  | printf("\nenter data after which node is to be deleted :"); |
|  | scanf("%d",&val); |
|  | ptr=head; |
|  | while(ptr->data!=val) |
|  | ptr=ptr->next; |
|  | if(ptr->next==NULL) |
|  | { |
|  | printf("\ncannot delete!"); |
|  | } |
|  | else if(ptr->next->next==NULL) |
|  | { |
|  | ptr->next=NULL; |
|  | } |
|  | else |
|  | { |
|  | temp=ptr->next; |
|  | ptr->next=temp->next; |
|  | temp->next->prev=ptr; |
|  | free(temp); |
|  | printf("\nnode deleted!"); |
|  | } |
|  | } |
|  | void show() |
|  | { |
|  | struct node \*ptr; |
|  | printf("\nprinting values.....\n"); |
|  | ptr=head; |
|  | while(ptr!=NULL) |
|  | { |
|  | printf("%d\n",ptr->data); |
|  | ptr=ptr->next; |
|  | } |
|  | } |
|  | void search() |
|  | { |
|  | struct node \*ptr; |
|  | int item,i=0,flag; |
|  | ptr=head; |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\nempty list!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter item to be searched :"); |
|  | scanf("%d",&item); |
|  | while(ptr!=NULL) |
|  | { |
|  | if(ptr->data==item) |
|  | { |
|  | printf("\nitem found at location %d",i+1); |
|  | flag=0; |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | flag=1; |
|  | } |
|  | i++; |
|  | ptr=ptr->next; |
|  | } |
|  | if(flag==1) |
|  | { |
|  | printf("\nitem not found!"); |
|  | } |
|  | } |
|  | }; |

1. Program to implement circular queue.

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | #define MAXSIZE 5 |
|  | int cq[MAXSIZE],f=-1,r=-1; |
|  | void enque(int[],int); |
|  | void deque(int[]); |
|  | void display(int[]); |
|  | void main() |
|  | { |
|  | int ch,x; |
|  | do |
|  | { |
|  | printf("\nCircular Queue:\n1)insert\n2)delete\n3)dispaly\n4)exit"); |
|  | printf("\nenter your choice:"); |
|  | scanf("%d",&ch); |
|  | switch(ch) |
|  | { |
|  | case 1: |
|  | printf("\nenter the element to be inserted:"); |
|  | scanf("%d",&x); |
|  | enque(cq,x); |
|  | break; |
|  | case 2: |
|  | deque(cq); |
|  | break; |
|  | case 3: |
|  | display(cq); |
|  | break; |
|  | case 4: |
|  | break; |
|  | default: |
|  | printf("\ninvalid choice!"); |
|  | } |
|  | } |
|  | while(ch!=4); |
|  | } |
|  | void enque(int cq[],int x) |
|  | { |
|  | if((f==0 && r==MAXSIZE-1) || (f==r+1)) |
|  | { |
|  | printf("\nqueue is full"); |
|  | return; |
|  | } |
|  | else if(r==-1) |
|  | { |
|  | r++; |
|  | f++; |
|  | } |
|  | else if(r==MAXSIZE-1 && f>0) |
|  | { |
|  | r=0; |
|  | } |
|  | else |
|  | { |
|  | r++; |
|  | } |
|  | cq[r]=x; |
|  | } |
|  | void deque(int cq[]) |
|  | { |
|  | if(f==-1) |
|  | { |
|  | printf("\nqueue is empty"); |
|  | } |
|  | else if(f==r) |
|  | { |
|  | printf("\ndeleted element",cq[f]); |
|  | f=-1; |
|  | r=-1; |
|  | } |
|  | else |
|  | { |
|  | printf("\ndeleted element",cq[f]); |
|  | f++; |
|  | } |
|  | } |
|  | void display(int cq[]) |
|  | { |
|  | int i; |
|  | printf("\n"); |
|  | if(f>r) |
|  | { |
|  | for(i=f;i<MAXSIZE;i++) |
|  | { |
|  | printf("%d",cq[i]); |
|  | } |
|  | for(i=r;i<r;i++) |
|  | { |
|  | printf("%d",cq[i]); |
|  | } |
|  | } |
|  | else |
|  | { |
|  | for(i=f;i<=r;i++) |
|  | { |
|  | printf("%d",cq[i]); |
|  | } |
|  | } |
|  | } |

1. **Program to merge two sorted arrays .**

#include<stdio.h>

#include<conio.h>

void main()

{

int a[10],b[10],c[20],i=0,j=0,k=0,m,n;

clrscr();

printf("Enter the size of the first array\n");

scanf("%d",&m);

printf("Enter the elements in first array\n");

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

{

scanf("%d",&a[i]);

}

printf("Enter the size of the second array\n");

scanf("%d",&n);

printf("Enter the elements in second array\n");

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

{

scanf("%d",&b[i]);

}

i=0;

j=0;

while((i<m)&&(j<n))

{

if(a[i]<=b[j])

{

c[k]=a[i];

i++;

}

else

{

c[k]=b[j];

j++;

}

k++;

}

while(i<m)

{

c[k]=a[i];

i++;

k++;

}

while(j<n)

{

c[k]=b[j];

j++;

k++;

}

printf("The first array is\n");

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

{

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

printf("\t\n");

}

printf("The second array is\n");

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

{

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

printf("\t\n");

}

printf("Merged array is\n");

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

{

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

printf("\t");

}

getch();

}

1. **Kruskal’s algorithm using the Disjoint set data structure.**

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | struct disjointSet { |
|  | int parent[10]; |
|  | int rank[10]; |
|  | int n; |
|  | } |
|  | dis; |
|  |  |
|  | void makeset() |
|  | { |
|  | int i; |
|  | for(i=0;i<dis.n;i++) |
|  | dis.parent[i]=i; |
|  | dis.rank[i]=0; |
|  | } |
|  |  |
|  | void displayset() |
|  | { |
|  | int i; |
|  | printf("\nparent array\n"); |
|  | for(i=0;i<dis.n;i++) |
|  | { |
|  | printf("%d",dis.parent[i]); |
|  | } |
|  | printf("\nrank of array\n"); |
|  | for(i=0;i<dis.n;i++) |
|  | { |
|  | printf("%d",dis.rank[i]); |
|  | } |
|  | printf("\n"); |
|  | } |
|  |  |
|  | int find(int x) |
|  | { |
|  | if(dis.parent[x]!=x) |
|  | { |
|  | dis.parent[x]=find(dis.parent[x]); |
|  | } |
|  | return dis.parent[x]; |
|  | } |
|  |  |
|  | void Union(int x,int y) |
|  | { |
|  | int xset=find(x) , yset=find(y); |
|  | if(xset==yset) |
|  | return; |
|  | if(dis.rank[xset]<dis.rank[yset]) |
|  | { |
|  | dis.parent[xset]=yset; |
|  | dis.rank[xset]=-1; |
|  | } |
|  | else if(dis.rank[xset]>dis.rank[yset]) |
|  | { |
|  | dis.parent[yset]=xset; |
|  | dis.rank[yset]=-1; |
|  | } |
|  | else |
|  | { |
|  | dis.parent[yset]=xset; |
|  | dis.rank[xset]=dis.rank[xset]+1; |
|  | dis.rank[yset]=-1; |
|  | } |
|  | } |
|  | int main() |
|  | { |
|  | int x,y,n; |
|  | printf("\nenter number of elements :\n"); |
|  | scanf("%d",&dis.n); |
|  | makeset(); |
|  | int ch,w; |
|  | do{ |
|  | printf("\n1.UNION\n2.FIND \n3.DISPLAY"); |
|  | printf("\nenter choice :"); |
|  | scanf("%d",&ch); |
|  | switch(ch) |
|  | { |
|  | case 1: |
|  | printf("\nenter elements to perform union :"); |
|  | scanf("%d%d",&x,&y); |
|  | Union(x,y); |
|  | break; |
|  | case 2: |
|  | printf("\nenter elements to check if connected components :"); |
|  | scanf("%d%d",&x,&y); |
|  | if(find(x)==find(y)) |
|  | printf("\nconnected components !"); |
|  | else |
|  | printf("\nnoi connected components !"); |
|  | break; |
|  | case 3: |
|  | displayset(); |
|  | break; |
|  | } |
|  | printf("\n do you want to continue ?(1/0)"); |
|  | scanf("%d",&w); |
|  | } |
|  | while(w==1); |
|  | return 0; |
|  | } |

1. **Program to implement stack operations.**

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | int top=-1,s[n],n; |
|  | void push(); |
|  | void pop(); |
|  | void display(); |
|  |  |
|  | void main() |
|  | { |
|  | int choice; |
|  | while(1) |
|  | { |
|  | printf("\n enter the value of n: "); |
|  | scanf("%d",&n); |
|  | printf("\n\n1.Push \n2.Pop \n3.Display \n4.Exit"); |
|  | printf("\n\nEnter your choice(1-4):"); |
|  | scanf("%d",&choice); |
|  |  |
|  | switch(choice) |
|  | { |
|  | case 1: push(); |
|  | break; |
|  | case 2: pop(); |
|  | break; |
|  | case 3: display(); |
|  | break; |
|  | case 4: exit(0); |
|  |  |
|  | default: printf("\n!!Invalid Choice!!"); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void push() |
|  | { |
|  | int x; |
|  |  |
|  | if(top>=n-1) |
|  | { |
|  | printf("\nStack is overflow!!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nEnter element to push:"); |
|  | scanf("%d",&x); |
|  | top++; |
|  | s[top]=x; |
|  | } |
|  | } |
|  |  |
|  | void pop() |
|  | { |
|  | if(top<=-1) |
|  | { |
|  | printf("\nStack is underflow!!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nDeleted element is %d",s[top]); |
|  | top--; |
|  | } |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | int i; |
|  |  |
|  | if(top==-1) |
|  | { |
|  | printf("\nStack is empty!!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\n Elements in the Stack is...\n"); |
|  | for(i=top;i>=0;i--) |
|  | printf("\n%d",s[i]); |
|  | } |
|  | } |

1. **Program to implement depth first search.**

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | void DFS(int v); |
|  | typedef enum boolean{false,true} bool; |
|  | int n,a[10][10]; |
|  | bool visited[10]; |
|  |  |
|  | void main() |
|  | { |
|  | int i,j,v; |
|  | printf("\nenter the no. of nodes in the graph\n"); |
|  | scanf("%d",&n); |
|  | printf("\nenter the adjacency matrix \n"); |
|  | for(i=1;i<=n;i++) |
|  | { |
|  | for(j=1;j<=n;j++) |
|  | { |
|  | scanf("%d",&a[i][j]); |
|  | } |
|  | } |
|  | printf("\nthe adjacency matrix shown as\n"); |
|  | for(i=1;i<=n;i++) |
|  | { |
|  | for(j=1;j<=n;j++) |
|  | { |
|  | printf("%d",a[i][j]); |
|  | } |
|  | printf("\n"); |
|  | } |
|  | printf("\nenter the starting node for Depth First search\n"); |
|  | scanf("%d",&v); |
|  | for(i=1;i<=n;i++) |
|  | visited[i]=false; |
|  | DFS(v); |
|  | getch(); |
|  | } |
|  |  |
|  | void DFS(int v) |
|  | { |
|  | int i,stack[10],top=-1,pop; |
|  | top++; |
|  | stack[top]=v; |
|  | while(top>=0) |
|  | { |
|  | pop=stack[top]; |
|  | top--; |
|  | if(visited[pop]==false) |
|  | { |
|  | printf("%d",pop); |
|  | visited[pop]=true; |
|  | } |
|  | else |
|  | continue; |
|  | for(i=n;i>=1;i--) |
|  | { |
|  | if(a[pop][i]==1 && visited[i]==false) |
|  | { |
|  | top++; |
|  | stack[top]=i; |
|  | } |
|  | } |
|  | } |
|  | } |

1. **Program to perform binary search tree operations.**

|  |
| --- |
| #include <stdio.h> |
|  | #include<conio.h> |
|  | #include<stdlib.h> |
|  | struct btnode |
|  | { |
|  | int value; |
|  | struct btnode \*l; |
|  | struct btnode \*r; |
|  | } |
|  | \*root=NULL,\*temp=NULL,\*t2,\*t1; |
|  | void delete2(); |
|  | void insert(); |
|  | void delete1(); |
|  | void create(); |
|  | void search(struct btnode \*t); |
|  | void search1(struct btnode \*t,int data); |
|  | int smallest(struct btnode \*t); |
|  | int largest(struct btnode \*t); |
|  | void inorder(struct btnode \*t); |
|  | void preorder(struct btnode \*t); |
|  | void postorder(struct btnode \*t); |
|  | int flag=1; |
|  | void main() |
|  | { |
|  | int ch; |
|  | printf("\nBINARY SEARCH TREE OPERATIONS\n1.INSERT\n2.DELETE\n3.INORDER\n4.PREORDER\n5.POSTORDER\n6.EXIT\n"); |
|  | while(1) |
|  | { |
|  | printf("\nenter your choice :"); |
|  | scanf("%d",&ch); |
|  | switch(ch) |
|  | { |
|  | case 1: |
|  | insert(); |
|  | break; |
|  | case 2: |
|  | delete1(); |
|  | break; |
|  | case 3: |
|  | inorder(root); |
|  | break; |
|  | case 4: |
|  | preorder(root); |
|  | break; |
|  | case 5: |
|  | postorder(root); |
|  | break; |
|  | case 6: |
|  | exit(0); |
|  | default: |
|  | printf("\ninvalid choice!"); |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | void insert() |
|  | { |
|  | create(); |
|  | if(root==NULL) |
|  | root=temp; |
|  | else |
|  | search(root); |
|  | } |
|  | void create() |
|  | { |
|  | int data; |
|  | printf("\nenter data to be inserted:"); |
|  | scanf("%d",&data); |
|  | temp=(struct btnode\*)malloc(1\*sizeof(struct btnode)); |
|  | temp->value=data; |
|  | temp->l=temp->r=NULL; |
|  | } |
|  | void search(struct btnode \*t) |
|  | { |
|  | if((temp->value>t->value)&&(t->r!=NULL)) |
|  | search(t->r); |
|  | else if((temp->value>t->value)&&(t->r==NULL)) |
|  | t->r=temp; |
|  | else if((temp->value<t->value)&&(t->l!=NULL)) |
|  | search(t->l); |
|  | else if((temp->value<t->value)&&(t->l==NULL)) |
|  | t->l=temp; |
|  | } |
|  | void inorder(struct btnode \*t) |
|  | { |
|  | if(root==NULL) |
|  | { |
|  | printf("\nno elements to display!"); |
|  | return; |
|  | } |
|  | if(t->l!=NULL) |
|  | inorder(t->l); |
|  | printf("%d->",t->value); |
|  | if(t->r!=NULL) |
|  | inorder(t->r); |
|  | } |
|  | void delete1() |
|  | { |
|  | int data; |
|  | if(root==NULL) |
|  | { |
|  | printf("\no elements!"); |
|  | return; |
|  | } |
|  | printf("\nenter the data to be deleted :"); |
|  | scanf("%d",&data); |
|  | t1=root; |
|  | t2=root; |
|  | search1(root,data); |
|  | } |
|  | void preorder(struct btnode \*t) |
|  | { |
|  | if(root==NULL) |
|  | { |
|  | printf("\nno elements to dislpay!"); |
|  | return; |
|  | } |
|  | printf("%d->",t->value); |
|  | if(t->l!=NULL) |
|  | preorder(t->l); |
|  | if(t->r!=NULL) |
|  | preorder(t->r); |
|  | } |
|  | void postorder(struct btnode \*t) |
|  | { |
|  | if(root==NULL) |
|  | { |
|  | printf("\nnoelements to display!"); |
|  | return; |
|  | } |
|  | if(t->l!=NULL) |
|  | postorder(t->l); |
|  | if(t->r!=NULL) |
|  | postorder(t->r); |
|  | printf("%d->",t->value); |
|  | } |
|  | void search1(struct btnode \*t,int data) |
|  | { |
|  | if((data>t->value)) |
|  | { |
|  | t1=t; |
|  | search1(t->r,data); |
|  | } |
|  | else if((data<t->value)) |
|  | { |
|  | t1=t; |
|  | search1(t->l,data); |
|  | } |
|  | else if((data==t->value)) |
|  | { |
|  | delete2(t); |
|  | } |
|  | } |
|  | void delete2(struct btnode \*t) |
|  | { |
|  | int k; |
|  | if((t->l==NULL)&&(t->r==NULL)) |
|  | { |
|  | if(t1->l==t) |
|  | { |
|  | t1->l=NULL; |
|  | } |
|  | else |
|  | { |
|  | t1->r=NULL; |
|  | } |
|  | t=NULL; |
|  | free(t); |
|  | return; |
|  | } |
|  | else if((t->r==NULL)) |
|  | { |
|  | if(t1==t) |
|  | { |
|  | root=t->l; |
|  | t1=root; |
|  | } |
|  | else if(t1->l==t) |
|  | { |
|  | t1->l=t->l; |
|  | } |
|  | else |
|  | { |
|  | t1->r=t->l; |
|  | } |
|  | t=NULL; |
|  | free(t); |
|  | return; |
|  | } |
|  | else if(t->l==NULL) |
|  | { |
|  | if(t1==t) |
|  | { |
|  | root=t->r; |
|  | t1=root; |
|  | } |
|  | else if(t1->r==t) |
|  | t1->r=t->r; |
|  | else |
|  | t1->l=t->r; |
|  | t=NULL; |
|  | free(t); |
|  | return; |
|  | } |
|  | else if((t->l!=NULL)&&(t->r!=NULL)) |
|  | { |
|  | t2=root; |
|  | if(t->r!=NULL) |
|  | { |
|  | k=smallest(t->r); |
|  | flag=1; |
|  | } |
|  | Else |
|  | { |
|  | k=largest(t->l); |
|  | flag=2; |
|  | } |
|  | search1(root,k); |
|  | t->value=k; |
|  | } |
|  | } |
|  | int smallest(struct btnode \*t) |
|  | { |
|  | t2=t; |
|  | if(t->l!=NULL) |
|  | { |
|  | t2=t; |
|  | return(smallest(t->l)); |
|  | } |
|  | Else |
|  | return(t->value); |
|  | } |
|  | int largest(struct btnode \*t) |
|  | { |
|  | if(t->r!=NULL) |
|  | { |
|  | t2=t; |
|  | return(largest(t->r)); |
|  | } |
|  | Else |
|  | return(t->value); |
|  | } |

1. **Program to perform linked stack operation.**

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | #include<stdlib.h> |
|  | void push(); |
|  | void pop(); |
|  | void display(); |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  | struct node \*head=NULL; |
|  | void main() |
|  | { |
|  | int ch=0; |
|  | do |
|  | { |
|  | printf("\n!!!LINKED STACK OPERATIONS!!!\n"); |
|  | printf("\n1)PUSH\n2)POP\n3)DISPLAY\n4)EXIT\n"); |
|  | printf("\nenter your choice ?\n"); |
|  | scanf("%d",&ch); |
|  | switch(ch) |
|  | { |
|  | case 1: |
|  | { |
|  | push(); |
|  | break; |
|  | } |
|  | case 2: |
|  | { |
|  | pop(); |
|  | break; |
|  | } |
|  | case 3: |
|  | { |
|  | display(); |
|  | break; |
|  | } |
|  | case 4: |
|  | { |
|  | exit(0); |
|  | } |
|  | default: |
|  | { |
|  | printf("\ninvalid choice!\n"); |
|  | break; |
|  | } |
|  | } |
|  | }while(ch!=4); |
|  | } |
|  | void push() |
|  | { |
|  | int item; |
|  | struct node \*ptr; |
|  | ptr=(struct node\*)malloc(sizeof(struct node)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow condition!\n"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter value :\n"); |
|  | scanf("%d",&item); |
|  | if(head==NULL) |
|  | { |
|  | ptr->data=item; |
|  | ptr->next=NULL; |
|  | head=ptr; |
|  | } |
|  | else |
|  | { |
|  | ptr->data=item; |
|  | ptr->next=head; |
|  | head=ptr; |
|  | } |
|  | printf("\nitem is pushed!\n"); |
|  | } |
|  | } |
|  | void pop() |
|  | { |
|  | struct node \*ptr; |
|  | if(head==NULL) |
|  | { |
|  | printf("\nunderflow condition!\n"); |
|  | } |
|  | else |
|  | { |
|  | ptr=head; |
|  | head=ptr->next; |
|  | free(ptr); |
|  | printf("\nitem is popped!\n"); |
|  | } |
|  | } |
|  | void display() |
|  | { |
|  | struct node \*ptr; |
|  | ptr=head; |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\nempty stack!\n"); |
|  | } |
|  | else |
|  | { |
|  | printf("\n\printing values...\n"); |
|  | while(ptr!=NULL) |
|  | { |
|  | printf("\n%d\n",ptr->data); |
|  | ptr=ptr->next; |
|  | } |
|  | } |
|  | } |

1. **Program to implement breadth first search.**

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1; |
|  |  |
|  | void BFS(int v) |
|  | { |
|  | for (i=1;i<=n;i++) |
|  | if(a[v][i] && !visited[i]) |
|  | q[++r]=i; |
|  | if(f<=r) |
|  | { |
|  | visited[q[f]]=1; |
|  | BFS(q[f++]); |
|  | } |
|  | } |
|  |  |
|  | void main() |
|  | { |
|  | int v; |
|  | printf("\nenter the number of vertices :"); |
|  | scanf("%d",&n); |
|  | for (i=1;i<=n;i++) |
|  | { |
|  | q[i]=0; |
|  | visited[i]=0; |
|  | } |
|  | printf("\nenter graph data in matrix form :\n"); |
|  | for (i=1;i<=n;i++) |
|  | for (j=1;j<=n;j++) |
|  | scanf("%d",&a[i][j]); |
|  | printf("\nenter the starting vertex :"); |
|  | scanf("%d",&v); |
|  | BFS(v); |
|  | printf("\nthe node which are reachable are :\n"); |
|  | for (i=1;i<=n;i++) |
|  | if(visited[i]) |
|  | printf("%d\t",i); |
|  | else |
|  | { |
|  | printf("\nBFS is not possible !"); |
|  | } |
|  | getch(); |
|  | } |

1. **Program to singly linked list operations.**

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  | #include<stdlib.h> |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  | struct node \*head=NULL; |
|  |  |
|  | void create(); |
|  | void insert\_beg(); |
|  | void insert\_pos(); |
|  | void insert\_end(); |
|  | void delete\_beg(); |
|  | void delete\_pos(); |
|  | void delete\_end(); |
|  | void delete\_all(); |
|  | void search(); |
|  | void display(); |
|  | void main() |
|  | { |
|  | int ch; |
|  | do |
|  | { |
|  | printf("\n!!!OPERATIONS ON SINGLY LINKEDLISTS!!!\n"); |
|  | printf("\n1.Create\n2.Insert a node at the begining\n3.Insert a node at specified position\n4.Insert a node at the end\n5.Delete a node from the begining\n6.Delete a node from specified position\n7.Delete a node from the end\n8.Delete the entire linklist\n9.Search for an element\n10.Display the linklist\n11.Exit\n"); |
|  | printf("\nenter your choice:"); |
|  | scanf("%d",&ch); |
|  | switch(ch) |
|  | { |
|  | case 1: |
|  | create(); |
|  | break; |
|  | case 2: |
|  | insert\_beg(); |
|  | break; |
|  | case 3: |
|  | insert\_pos(); |
|  | break; |
|  | case 4: |
|  | insert\_end(); |
|  | break; |
|  | case 5: |
|  | delete\_beg(); |
|  | break; |
|  | case 6: |
|  | delete\_pos(); |
|  | break; |
|  | case 7: |
|  | delete\_end(); |
|  | break; |
|  | case 8: |
|  | delete\_all(); |
|  | break; |
|  | case 9: |
|  | search(); |
|  | break; |
|  | case 10: |
|  | display(); |
|  | break; |
|  | case 11: |
|  | exit(0); |
|  | break; |
|  | default: |
|  | printf("\ninvalid choice!"); |
|  | break; |
|  | } |
|  | } |
|  | while(ch!=11); |
|  | } |
|  | void create() |
|  | { |
|  | struct node \*temp,\*ptr; |
|  | temp=(struct node \*)malloc(sizeof(struct node)); |
|  | if(temp==NULL) |
|  | { |
|  | printf("\nmemory cannot be allocated\n"); |
|  | } |
|  | printf("\nenter the value:"); |
|  | scanf("%d",&temp->data); |
|  | temp->next=NULL; |
|  | if(head==NULL) |
|  | { |
|  | head=temp; |
|  | } |
|  | else |
|  | { |
|  | ptr=head; |
|  | while(ptr->next!=NULL) |
|  | { |
|  | ptr=ptr->next; |
|  | } |
|  | ptr->next=temp; |
|  | } |
|  | } |
|  | void insert\_beg() |
|  | { |
|  | struct node \*ptr; |
|  | int item; |
|  | ptr=(struct node\*)malloc(sizeof(struct node\*)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter the value:"); |
|  | scanf("%d",&item); |
|  | ptr->data=item; |
|  | ptr->next=head; |
|  | head=ptr; |
|  | printf("\nnode is inserted!"); |
|  | } |
|  | } |
|  | void insert\_pos() |
|  | { |
|  | int i,loc,item; |
|  | struct node \*ptr,\*temp; |
|  | ptr=(struct node\*)malloc(sizeof(struct node)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow!"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter the value:"); |
|  | scanf("%d",&item); |
|  | ptr->data=item; |
|  | printf("\nenter the location after which you want to insert:"); |
|  | scanf("%d",&loc); |
|  | temp=head; |
|  | for(i=0;i<loc;i++) |
|  | { |
|  | temp=temp->next; |
|  | if(temp==NULL) |
|  | { |
|  | printf("\nnode can't be inserted!"); |
|  | return; |
|  | } |
|  | } |
|  | ptr->next=temp->next; |
|  | temp->next=ptr; |
|  | printf("\nnode is inserted!"); |
|  | } |
|  | } |
|  | void insert\_end() |
|  | { |
|  | struct node \*ptr,\*temp; |
|  | int item; |
|  | ptr=(struct node\*)malloc(sizeof(struct node)); |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\noverflow"); |
|  | } |
|  | else |
|  | { |
|  | printf("\nenter the value:"); |
|  | scanf("%d",&item); |
|  | ptr->data=item; |
|  | if(head==NULL) |
|  | { |
|  | ptr->next=NULL; |
|  | head=ptr; |
|  | printf("\nnode inserted!"); |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | temp->next=ptr; |
|  | ptr->next=NULL; |
|  | printf("\nnode inserted!"); |
|  | } |
|  | } |
|  | } |
|  | void delete\_beg() |
|  | { |
|  | struct node \*ptr; |
|  | if(head==NULL) |
|  | { |
|  | printf("\nlist is empty!"); |
|  | } |
|  | else |
|  | { |
|  | ptr=head; |
|  | head=ptr->next; |
|  | free(ptr); |
|  | printf("\nnode deleted from beginning!"); |
|  | } |
|  | } |
|  | void delete\_pos() |
|  | { |
|  | struct node \*ptr,\*ptr1; |
|  | int loc,i; |
|  | printf("\nenter location of node after which deletion is to be performed:"); |
|  | scanf("%d",&loc); |
|  | ptr=head; |
|  | for(i=0;i<loc;i++) |
|  | { |
|  | ptr1=ptr; |
|  | ptr=ptr->next; |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\ncan't delete!"); |
|  | return; |
|  | } |
|  | } |
|  | ptr1->next=ptr->next; |
|  | free(ptr); |
|  | printf("\ndeleted node %d",loc+1); |
|  | } |
|  | void delete\_end() |
|  | { |
|  | struct node \*ptr,\*ptr1; |
|  | if(head==NULL) |
|  | { |
|  | printf("\nlist is empty!"); |
|  | } |
|  | else if(head->next==NULL) |
|  | { |
|  | head=NULL; |
|  | free(head); |
|  | printf("\nonly node of the list is deleted"); |
|  | } |
|  | else |
|  | { |
|  | ptr=head; |
|  | while(ptr->next!=NULL) |
|  | { |
|  | ptr1=ptr; |
|  | ptr=ptr->next; |
|  | } |
|  | ptr1->next=NULL; |
|  | free(ptr); |
|  | printf("\ndeleted node from the last!"); |
|  | } |
|  | } |
|  | void delete\_all() |
|  | { |
|  | struct node \*temp; |
|  | while(head!=NULL) |
|  | { |
|  | temp=head; |
|  | head=head->next; |
|  | free(temp); |
|  | } |
|  | printf("\nsuccessfully deleted entire nodes of linked list!"); |
|  | } |
|  | void search() |
|  | { |
|  | struct node \*ptr; |
|  | int item,i=0,flag=0; |
|  | ptr=head; |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\nempty list!"); |
|  | } |
|  | Else |
|  | { |
|  | printf("\nenter item to be searched?\n"); |
|  | scanf("%d",&item); |
|  | while(ptr!=NULL) |
|  | { |
|  | if(ptr->data==item) |
|  | { |
|  | printf("\nitem found at location %d",i+1); |
|  | flag=0; |
|  | } |
|  | Else |
|  | { |
|  | flag=1; |
|  | } |
|  | i++; |
|  | ptr=ptr->next; |
|  | } |
|  | if(flag==1) |
|  | { |
|  | printf("\nitem not found!"); |
|  | } |
|  | } |
|  | } |
|  | void display() |
|  | { |
|  | struct node \*ptr; |
|  | ptr=head; |
|  | if(ptr==NULL) |
|  | { |
|  | printf("\nno dataitems found so nothing to print!"); |
|  | } |
|  | Else |
|  | { |
|  | printf("\nprinting values ...\n"); |
|  | while(ptr!=NULL) |
|  | { |
|  | printf("\n%d",ptr->data); |
|  | ptr=ptr->next; |
|  | } |
|  | } |
|  | } |