**/\*1.a C++ program to implement stack ADT using array\*/**

**#include<iostream.h>**

**#include<stdlib.h>**

**#include<conio.h>**

**template <class T>**

**class stack**

**{**

**T \*a;**

**int top;**

**int n;**

**public:**

**stack();**

**void push();**

**void pop();**

**void display();**

**};**

**template <class T>**

**stack<T>::stack()**

**{**

**top = -1;**

**a = new T[20];**

**}**

**template <class T>**

**void stack<T>::push()**

**{**

**int n;**

**if(top >= 20)**

**{**

**cout<<"The stack is full\n";**

**}**

**else**

**{**

**cout<<"Enter the element:\n";**

**cin>>n;**

**a[++top]= n;**

**}**

**}**

**template<class T>**

**void stack <T>::pop()**

**{**

**if(top <= 0)**

**{**

**cout<<"The stack is empty\n";**

**}**

**else**

**{**

**n = a[top--];**

**cout<<"The deleted element is:"<<n<<"\n";**

**}**

**}**

**template<class T>**

**void stack<T>::display()**

**{**

**cout<<"The stack elements are:";**

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

**{**

**cout<<a[i]<<" \n";**

**}**

**}**

**void main()**

**{**

**stack<int> stk;**

**int ch=0;**

**clrscr();**

**do**

**{**

**cout<<"1.Push 2.Pop 3.Display 4.Exit\n";**

**cout<<"Enter you choice:\n";**

**cin>>ch;**

**switch(ch)**

**{**

**case 1:stk.push();**

**break;**

**case 2:stk.pop();**

**break;**

**case 3:stk.display();**

**break;**

**case 4:exit(0);**

**default:cout<<"Enter the correct option\n";**

**}**

**}while(1);**

**}**

**OUTPUT:**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**1**

**Enter the element:**

**7**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**1**

**Enter the element:**

**3**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**1**

**Enter the element:**

**5**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**3**

**5 3 7**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**2**

**The deleted element is: 5**

**Enter your choice:**

**3**

**3 7**

**1.Push 2.Pop 3.Display 4.Exit**

**4**

**/\* 1.b C++ program to implement queue ADT using array\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**class queue**

**{**

**int q[20];**

**int rear,front;**

**public:**

**queue()**

**{**

**rear=-1;**

**front=-1;**

**}**

**void insert(int);**

**void delet();**

**void display();**

**};**

**void queue::insert(int x)**

**{**

**if(rear > 19)**

**{**

**cout <<"queue over flow";**

**front=rear=-1;**

**return;**

**}**

**q[++rear]=x;**

**}**

**void queue::delet()**

**{**

**if(front==rear)**

**{**

**cout <<"queue under flow";**

**return;**

**}**

**cout <<"deleted" <<q[++front];**

**}**

**void queue::display()**

**{**

**if(rear==front)**

**{**

**cout <<" queue empty";**

**return;**

**}**

**for(int i=front+1;i<=rear;i++)**

**cout <<q[i]<<" ";**

**}**

**void main()**

**{**

**int ch;**

**queue qu;**

**while(1)**

**{**

**cout <<"\n1.insert 2.delet 3.display 4.exit\nEnter your choice";**

**cin >> ch;**

**switch(ch)**

**{**

**case 1:cout <<"enter the element";**

**cin >> ch;**

**qu.insert(ch);**

**break;**

**case 2:qu.delet();**

**break;**

**case 3:qu.display();**

**break;**

**case 4: exit(0);**

**}**

**}**

**}**

**OUTPUT:**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 21**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 33**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 41**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 53**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**3**

**21 33 41 53**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**2**

**Deleted 21**

**Enter your choice**

**3**

**33 41 53**

**/\*2.a C++ program to implement stack ADT using singly linked list.\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#include<alloc.h>**

**int flag=0;**

**struct node**

**{**

**int value;**

**struct node\* next;**

**};**

**class stack**

**{**

**node\* top,\*base;**

**public :**

**void init()**

**{**

**base->next=NULL;**

**}**

**void push(int num);**

**void pop();**

**void display();**

**};**

**void stack::push(int num)**

**{**

**node\* data;**

**if(flag==0)**

**{**

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

**data->value=num;**

**data->next=base;**

**top=data;**

**flag=1;**

**}**

**else**

**{**

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

**data->value=num;**

**data->next=top;**

**top=data;**

**}**

**}**

**void stack::pop()**

**{**

**if(top->next!=NULL)**

**{**

**cout<<"\n\nThe deleted element is "<<top->value;**

**top=top->next;**

**}**

**else**

**{**

**cout<<"\n\nStack empty";**

**flag=0;**

**}**

**}**

**void stack::display()**

**{**

**node\* ptr;**

**ptr=top;**

**while(ptr->next!=NULL)**

**{**

**cout<<ptr->value<<"\n";**

**ptr=ptr->next;**

**}**

**}**

**Void main()**

**{**

**int choice,num,n;**

**class stack s1;**

**clrscr();**

**s1.init();**

**do**

**{**

**cout<<"\n1.Push\n2.Pop\n3.Display\n4.Exit\n\n ";**

**cout<<"Enter your choice : ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1 : cout<<"\nEnter the element : ";**

**cin>>num;**

**s1.push(num);**

**break;**

**case 2 : s1.pop();**

**break;**

**case 3 : s1.display();**

**break;**

**case 4 : exit(0);**

**default : cout<<"\n\nInvalid choice...Enter…again";**

**}**

**}while(1);**

**}**

**OUTPUT:**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**1**

**Enter the element:**

**7**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**1**

**Enter the element:**

**3**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**1**

**Enter the element:**

**5**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**3**

**5 3 7**

**1.Push 2.Pop 3.Display 4.Exit**

**Enter your choice:**

**2**

**The deleted element is: 5**

**Enter your choice:**

**3**

**3 7**

**1.Push 2.Pop 3.Display 4.Exit**

**4**

**/\* 2.b C++ program to implement queue ADT using singly linked list.\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**int flag=0;**

**template <class T>**

**class queue**

**{**

**struct node**

**{**

**T data;**

**node \*next;**

**}\*first,\*last;**

**public :**

**void insert(T n);**

**void del();**

**void display();**

**};**

**template <class T>**

**void queue<T>::insert(T n)**

**{**

**node \*temp;**

**temp=new node;**

**temp->data=n;**

**if(flag==0)**

**{**

**first=temp;**

**first->next=NULL;**

**last=first;**

**flag=1;**

**}**

**else if(flag==1)**

**{**

**last->next=temp;**

**last=temp;**

**last->next=NULL;**

**}**

**}**

**template <class T>**

**void queue<T>::del()**

**{**

**if(first==NULL)**

**{**

**cout<<"\nEmpty queue";**

**flag=0;**

**}**

**else**

**{**

**cout<<"\nDeleted element is : "<<first->data;**

**first=first->next;**

**}**

**}**

**template <class T>**

**void queue<T>::display()**

**{**

**node \*temp;**

**temp=first;**

**if(first==NULL)**

**cout<<"\nEmpty queue";**

**else**

**{**

**cout<<"\nElements of the queue \n\n";**

**do**

**{**

**cout<<temp->data<<endl;**

**temp=temp->next;**

**}while(temp!=NULL);**

**}**

**}**

**int main()**

**{**

**int choice;**

**queue<int> q1;**

**int element;**

**clrscr();**

**do**

**{**

**cout<<"\n1.Insert\n2.Delete\n3.Display\n4.Exit\n\nEnter your choice : ";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1 : cout<<"\nEnter the element : ";**

**cin>>element;**

**q1.insert(element);**

**break;**

**case 2 : q1.del();**

**break;**

**case 3 : q1.display();**

**break;**

**default : exit(0);**

**}**

**}while(1);**

**}**

**OUTPUT:**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 21**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 33**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 41**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**1**

**Enter the element 53**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**3**

**21 33 41 53**

**1.Insert 2.Delete 3.Display 4.Exit**

**Enter your choice**

**2**

**Deleted 21**

**Enter your choice**

**3**

**33 41 53**

**/\*3.C++ Program to implement deque ADT using a doubly linked list\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#define NULL 0**

**void ins\_beg();**

**void ins\_end();**

**void del\_beg();**

**void del\_end();**

**void traverse();**

**struct stack**

**{**

**struct stack \*prev;**

**int n;**

**struct stack \*next;**

**}\*head=NULL;**

**typedef struct stack node;**

**void main()**

**{**

**int choice;**

**clrscr();**

**do**

**{**

**cout<<"\nEnter choice";**

**cout<<"\n1.insertion at beginning \n2.insertion at end\n";**

**cout<<"3.deletion at beginning\n";**

**cout<<"4.deletion at end \n5.display\n6.exit\n";**

**cin>>choice;**

**switch (choice)**

**{**

**case 1:ins\_beg();**

**break;**

**case 2:ins\_end();**

**break;**

**case 3:del\_beg();**

**break;**

**case 4:del\_end();**

**break;**

**case 5:traverse();**

**break;**

**default:exit (0);**

**}**

**}while(1);**

**}**

**void ins\_beg()**

**{**

**node \*p;**

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

**cout<<"Enter element to be inserted"<<endl;**

**cin>>(p->n);**

**p->prev=NULL;**

**p->next=head;**

**head=p;**

**}**

**void ins\_end()**

**{**

**node \*p,\*q=head;**

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

**while(q->next!=NULL)**

**{**

**q=q->next;**

**}**

**q->next=p;**

**p->prev=q;**

**p->next=NULL;**

**cout<<"enter element to be inserted "<<endl;**

**cin>>(p->n);**

**}**

**void del\_beg()**

**{**

**node \*p=head;**

**head=head->next;**

**cout<<"deleted element is "<<(p->n);**

**head->prev=NULL;**

**free(p);**

**}**

**void del\_end()**

**{**

**node \*p=head,\*q;**

**if((p->next)==NULL)**

**{**

**cout<<"delted element is "<<p->n;**

**free(p);**

**head=NULL;**

**}**

**else**

**{**

**while((p->next)->next!=NULL)**

**{**

**p=p->next;**

**}**

**cout<<"deleted element is"<<(p->next)->n;**

**q=p;**

**q->next=NULL;**

**p=q->next;**

**free(p);**

**}**

**}**

**void traverse()**

**{**

**node \*p=head;**

**while(p!=NULL)**

**{**

**cout<<p->n<<" ";**

**p=p->next;**

**}**

**}**

**OUTPUT**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**1**

**Enter element to be inserted:2**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**1**

**Enter element to be inserted:1**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**2**

**Enter element to be inserted:3**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**2**

**Enter element to be inserted:4**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**5**

**1 2 3 4**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**3**

**delted element is 1**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**4**

**delted element is 4**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**5**

**2 3**

**6.exit**

**Enter choice**

**1.insertion at beginning**

**2.insertion at end**

**3.deletion at beginning**

**4.deletion at end**

**5.display**

**6.exit**

**6**

**/\*4.C++ Program For Implementation Of Binary Search Tree and perform insertion deletion,searching,display of tree.\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**class bintree**

**{**

**typedef struct bst**

**{**

**int data;**

**struct bst \*left,\*right;**

**}node;**

**node \*root,\*New,\*temp,\*parent;**

**public:**

**bintree()**

**{**

**root=NULL;**

**}**

**void create();**

**void display();**

**void delet();**

**void find();**

**void insert(node \*,node \*);**

**void inorder(node \*);**

**void search(node \*\*,int,node \*\*);**

**void del(node \*,int);**

**};**

**void bintree::create()**

**{**

**New=new node;**

**New->left=NULL;**

**New->right=NULL;**

**cout<<"\n Enter The Element ";**

**cin>>New->data;**

**if(root==NULL)**

**root=New;**

**else**

**insert(root,New);**

**}**

**void bintree::insert(node \*root,node \*New)**

**{**

**if(New->data<root->data)**

**{**

**if(root->left==NULL)**

**root->left=New;**

**else**

**insert(root->left,New);**

**}**

**if(New->data>root->data)**

**{**

**if(root->right==NULL)**

**root->right=New;**

**else**

**insert(root->right,New);**

**}**

**}**

**void bintree::display()**

**{**

**if(root==NULL)**

**cout<<"Tree Is Not Created";**

**else**

**{**

**cout<<"\n The Tree is : ";**

**inorder(root);**

**}**

**}**

**void bintree::inorder(node \*temp)**

**{**

**if(temp!=NULL)**

**{**

**inorder(temp->left);**

**cout<<" "<<temp->data;**

**inorder(temp->right);**

**}**

**}**

**void bintree::find()**

**{**

**int key;**

**cout<<"\n Enter The Element Which You Want To Search ";**

**cin>>key;**

**temp=root;**

**search(&temp,key,&parent);**

**if(temp==NULL)**

**cout<<"\n Element is not present ";**

**else**

**cout<<"\nParent of node "<<temp->data<<" is "<<parent->data;**

**}**

**void bintree::search(node \*\*temp,int key,node \*\*parent)**

**{**

**if(\*temp==NULL)**

**cout<<endl<<"Tree is Not Created"<<endl;**

**else**

**{**

**while(\*temp!=NULL)**

**{**

**if((\*temp)->data==key)**

**{**

**cout<<"\nThe element "<<(\*temp)->data<<" is Present";**

**break;**

**}**

**\*parent=\*temp;**

**if((\*temp)->data>key)**

**\*temp=(\*temp)->left;**

**else**

**\*temp=(\*temp)->right;**

**}**

**}**

**return;**

**}**

**void bintree::delet()**

**{**

**int key;**

**cout<<"\n Enter The Element U wish to Delete";**

**cin>>key;**

**if(key==root->data)**

**{**

**bintree();**

**}**

**else**

**del(root,key);**

**}**

**void bintree::del(node \*root,int key)**

**{**

**node \*temp\_succ;**

**if(root==NULL)**

**cout<<"Tree is not Created!";**

**else**

**{**

**temp=root;**

**search(&temp,key,&parent);**

**if(temp->left!=NULL && temp->right!=NULL)**

**{**

**parent=temp;**

**temp\_succ=temp->right;**

**while(temp\_succ->left!=NULL)**

**{**

**parent=temp\_succ;**

**temp\_succ=temp\_succ->left;**

**}**

**temp->data=temp\_succ->data;**

**temp->right=NULL;**

**cout<<" Now Deleted it!";**

**return;**

**}**

**if(temp->left!=NULL && temp->right==NULL)**

**{**

**if(parent->left==temp)**

**parent->left=temp->left;**

**else**

**parent->right=temp->left;**

**temp=NULL;**

**delete temp;**

**cout<<" Now Deleted it!";**

**return;**

**}**

**if(temp->left==NULL && temp->right!=NULL)**

**{**

**if(parent->left==temp)**

**parent->left=temp->right;**

**else**

**parent->right=temp->right;**

**temp=NULL;**

**delete temp;**

**cout<<" Now Deleted it!";**

**return;**

**}**

**if(temp->left==NULL && temp->right==NULL)**

**{**

**if(parent->left==temp)**

**parent->left=NULL;**

**else**

**parent->right=NULL;**

**cout<<" Now Deleted it!";**

**return;**

**}**

**}**

**}**

**void main()**

**{**

**int choice;**

**char ans='N';**

**bintree tr;**

**clrscr();**

**cout<<"\n\t Program For Binary Search Tree ";**

**do**

**{**

**cout<<"\n1.Create\n2.Search\n3.Delete\n4.Display";**

**cout<<"\n Enter your choice :";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:do**

**{**

**tr.create();**

**cout<<"Do u Want To enter More elements?(y/n)"<<endl;**

**ans=getche();**

**}while(ans=='y');**

**break;**

**case 2:tr.find();**

**break;**

**case 3:tr.delet();**

**break;**

**case 4:tr.display();**

**break;**

**default:exit(0);**

**}**

**}while(1);**

**}**

**OUTPUT**

**Enter choice**

**1.create 2.search 3.delete 4.display 5.exit**

**1**

**Enter The Element:10**

**Do u Want To enter More elements?(y/n):y**

**Enter The Element:8**

**Do u Want To enter More elements?(y/n):y**

**Enter The Element:7**

**Do u Want To enter More elements?(y/n):y**

**Enter The Element:9**

**Do u Want To enter More elements?(y/n):y**

**Enter The Element:12**

**Do u Want To enter More elements?(y/n):y**

**Enter The Element:11**

**Do u Want To enter More elements?(y/n):y**

**Enter The Element:13**

**Do u Want To enter More elements?(y/n):n**

**Enter choice**

**1.create 2.search 3.delete 4.display 5.exit**

**4**

**7 8 9 10 11 12 13**

**Enter choice**

**1.create 2.search 3.delete 4.display 5.exit**

**2**

**Enter The Element Which You Want To Search**

**13**

**Parent node of 13 is 12**

**Enter choice**

**1.create 2.search 3.delete 4.display 5.exit**

**3**

**Enter The Element to Delete**

**12**

**Now Deleted it!**

**Enter choice**

**1.create 2.search 3.delete 4.display 5.exit**

**4**

**7 8 9 10 11 13**

**Enter choice**

**1.create 2.search 3.delete 4.display 5.exit**

**5**

**/\*5.C++ Program that uses non-recursive functions to traverse binary search tree in Preorder, Inorder,Postorder\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#define NULL 0**

**class bst**

**{**

**public:**

**struct binary**

**{**

**int n;**

**struct binary \*left;**

**struct binary \*right;**

**};**

**typedef struct binary node;**

**node \*root,\*temp,\*parent,\*New,\*current,\*s[10];**

**bst()**

**{**

**root=NULL;**

**}**

**void create();**

**void display();**

**void insert(node \*,node \*);**

**void inorder(node \*);**

**void preorder(node \*);**

**void postorder(node \*);**

**void push(node \*item,int \*top,node \*s[10]);**

**int stempty(int);**

**void pop(int \*top,node \*s[10],node\*\*current);**

**};**

**int bst::stempty(int top)**

**{**

**if(top==-1)**

**return 1;**

**else**

**return 0;**

**}**

**void bst::create()**

**{**

**New=new node;**

**New->left=NULL;**

**New->right=NULL;**

**cout<<"Enter element to be inserted\n";**

**cin>>New->n;**

**if(root==NULL)**

**root=New;**

**else**

**insert(root,New);**

**}**

**void bst::insert(node \*root,node \*New)**

**{**

**if((New->n)<(root->n))**

**{**

**if(root->left==NULL)**

**root->left=New;**

**else**

**insert(root->left,New);**

**}**

**if((New->n)>=(root->n))**

**{**

**if(root->right==NULL)**

**root->right=New;**

**else**

**insert(root->right,New);**

**}**

**}**

**void bst::push(node \*item,int \*top,node \*s[])**

**{**

**\*top=\*top+1;**

**s[\*top]=item;**

**}**

**void bst::pop(int \*top,node \*s[],node\*\*current)**

**{**

**\*current=s[(\*top)--];**

**}**

**void bst::inorder(node \*root)**

**{**

**int top=-1;**

**if(root==NULL)**

**{**

**cout<<"tree is empty\n";**

**return;**

**}**

**current=root;**

**for(;;)**

**{**

**while(current!=NULL)**

**{**

**push(current,&top,s);**

**current=current->left;**

**}**

**if(!stempty(top))**

**{**

**pop(&top,s,&current);**

**cout<<" "<<current->n;**

**current=current->right;**

**}**

**else**

**return;**

**}**

**}**

**void bst::preorder(node \*root)**

**{**

**int top=-1;**

**if(root==NULL)**

**{**

**cout<<"Tree is empty";**

**return;**

**}**

**current=root;**

**for(;;)**

**{**

**while(current!=NULL)**

**{**

**cout<<" "<<current->n;**

**push(current,&top,s);**

**current=current->left;**

**}**

**if(!stempty(top))**

**{**

**pop(&top,s,&current);**

**current=current->right;**

**}**

**else**

**return;**

**}**

**}**

**void bst::postorder(node \*root)**

**{**

**struct stack**

**{**

**node \*element;**

**int check;**

**}st[10];**

**int top=-1;**

**if(root==NULL)**

**{**

**cout<<"Tree is empty";**

**return;**

**}**

**current=root;**

**for(;;)**

**{**

**while(current!=NULL)**

**{**

**top++;**

**st[top].element=current;**

**st[top].check=1;**

**current=current->left;**

**}**

**while(st[top].check==0)**

**{**

**current=st[top].element;**

**top--;**

**cout<<" "<<current->n;**

**if(stempty(top))**

**return;**

**}**

**current=st[top].element;**

**current=current->right;**

**st[top].check=0;**

**}**

**}**

**void bst::display()**

**{**

**if(root==NULL)**

**cout<<"root not created";**

**else**

**{**

**cout<<"\nInorder traversal of Tree is \n";**

**inorder(root);**

**cout<<"\nPreorder traversal of Tree is \n";**

**preorder(root);**

**cout<<"\nPostorder traversal of Tree is \n";**

**postorder(root);**

**}**

**}**

**void main()**

**{**

**int choice;**

**clrscr();**

**bst tr;**

**do**

**{**

**cout<<" Enter choice\n ";**

**cout<<"1.create 2.display 3.exit";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:**

**tr.create();**

**break;**

**case 2:**

**tr.display();**

**break;**

**case 3:**

**exit(0);**

**}**

**}while(1);**

**}**

**OUTPUT**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:10**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:12**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:8**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:9**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:7**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:13**

**Enter choice**

**1.create 2.display 3.exit**

**1**

**Enter element to be inserted:11**

**Enter choice**

**1.create 2.display 3.exit**

**2**

**InInorder traversal of Tree is**

**7 8 9 10 11 12 3**

**Preorder traversal of Tree is**

**10 8 7 9 12 11 13**

**Postorder traversal of Tree is**

**7 9 8 11 13 12 10**

**Enter choice**

**1.create 2.display 3.exit**

**3**

**/\*6.a C++ Program for implementation of bfs of a graph\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**int g[10][10],i,j,k,n,q[10],front=-1,rear=-1,v1,v[10];**

**void main()**

**{**

**int m;**

**char c;**

**clrscr();**

**cout<<"Enter d for directed and u for undirected \n";**

**cin>>c;**

**cout<<"Enter number of vertices";**

**cin>>n;**

**cout<<"Enter number of edges";**

**cin>>m;**

**cout<<"\nEnter the Edges \n";**

**for(k=1;k<=m;k++)**

**{**

**cin>>i>>j;**

**g[i][j]=1;**

**if(c=='u'||c=='U')**

**g[j][i]=1;**

**}**

**cout<<"\nEnter vertex from which you want to traverse\n";**

**cin>>v1;**

**cout<<"Visitied vertices(bfs)\n";**

**v[v1]=1;**

**q[++rear]=v1;**

**while(front!=rear)**

**{**

**v1=q[++front];**

**cout<<v1<< " ";**

**for(j=1;j<=n;j++)**

**if(g[v1][j]==1 && v[j]!=1)**

**{**

**q[++rear]=j;**

**v[j]=1;**

**}**

**}**

**getch();**

**}**

**OUTPUT**

**Enter d for directed and u for undirected**

**u**

**Enter number of vertices**

**4**

**Enter number of edges**

**4**

**Enter the Edges**

**1 2**

**1 3**

**2 4**

**3 4**

**Enter vertex from which you want to traverse:1**

**Visitied vertices(bfs)**

**1 2 3 4**

**/\*6.b C++ Program for implementation of dfs of a graph\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**int g[10][10],i,j,k,n,st[10],top=-1,v1,v[10];**

**void main()**

**{**

**int m;**

**char c;**

**clrscr();**

**cout<<"Enter d for directed and u for undirected \n";**

**cin>>c;**

**cout<<"Enter number of vertices";**

**cin>>n;**

**cout<<"Enter number of edges";**

**cin>>m;**

**cout<<"\n Enter the Edges \n";**

**for(k=1;k<=m;k++)**

**{**

**cin>>i>>j;**

**g[i][j]=1;**

**if(c=='u'||c=='U')**

**g[j][i]=1;**

**}**

**cout<<"\nEnter vertex from which you want to traverse\n";**

**cin>>v1;**

**cout<<"Visitied vertices(dfs)\n";**

**v[v1]=1;**

**st[++top]=v1;**

**while(top!=-1)**

**{**

**v1=st[top--];**

**cout<<v1<<" ";**

**for(j=n;j>=1;j--)**

**if(g[v1][j]==1 && v[j]!=1)**

**{**

**st[++top]=j;**

**v[j]=1;**

**}**

**}**

**getch();**

**}**

**OUTPUT**

**Enter d for directed and u for undirected**

**u**

**Enter number of vertices**

**4**

**Enter number of edges**

**4**

**Enter the Edges**

**1 2**

**1 3**

**2 4**

**3 4**

**Enter vertex from which you want to traverse:2**

**Visitied vertices(dfs)**

**2 1 3 4**

**/\*7.aC++ Program for implementing Merge Sort\*/**

**#include<iostream.h>**

**#include <conio.h>**

**#include<stdlib.h>**

**template <class T>**

**class Merge**

**{**

**private:**

**T a[10],b[10];**

**int n;**

**public:**

**int get\_data();**

**void display\_data();**

**void combine(int,int,int);**

**void merge\_sort(int,int);**

**};**

**template <class T>**

**int Merge<T>::get\_data()**

**{**

**int i;**

**cout<<"\n Enter the length of list :";**

**cin>>n;**

**cout<<"\n Enter list elments :";**

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

**cin>>a[i];**

**return n;**

**}**

**template <class T>**

**void Merge<T>::display\_data()**

**{**

**int i;**

**cout<<"\n\n The Sorted Array Is ...\n";**

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

**cout<<" "<<a[i];**

**}**

**template <class T>**

**void Merge<T>::combine(int low,int mid,int high)**

**{**

**int l,i,j,k;**

**l=low;**

**i=low;**

**j=mid+1;**

**while(l <= mid && j <= high)**

**{**

**if(a[l]<=a[j])**

**b[i]=a[l++];**

**else**

**b[i]=a[j++];**

**i++;**

**}**

**if(l > mid )**

**for(k=j;k<=high;k++)**

**b[i++]=a[k];**

**else**

**for(k=l;k<=mid;k++)**

**b[i++]=a[k];**

**for(k=low;k<=high;k++)**

**a[k]=b[k];**

**}**

**template <class T>**

**void Merge<T>::merge\_sort(int low,int high)**

**{**

**int mid;**

**if(low < high)**

**{**

**mid = (low+high)/2;**

**merge\_sort(low,mid);**

**merge\_sort(mid+1,high);**

**combine(low,mid,high);**

**}**

**}**

**void main()**

**{**

**int low,high,n;**

**Merge <int> obj;**

**clrscr();**

**cout<<"\n Merge Sort \n";**

**n=obj.get\_data();**

**low = 0;**

**high = n - 1;**

**obj.merge\_sort(low,high);**

**obj.display\_data();**

**getch();**

**}**

**OUTPUT**

**Merge Sort**

**Enter the length of list:**

**5**

**Enter list elements:**

**77**

**11**

**32**

**43**

**56**

**The Sorted Array Is ...**

**11 32 43 56 77**

**/\*7.b C++ Program for implementing Heap Sort\*/**

**#include<iostream.h>**

**#include<stdlib.h>**

**#include<conio.h>**

**#define MAX 20**

**template <class T>**

**class Heap**

**{**

**private:**

**T arr[MAX];**

**int n;**

**public:**

**Heap();**

**void insert();**

**void makeheap();**

**void heapsort();**

**void display();**

**};**

**template <class T>**

**Heap<T>::Heap()**

**{**

**n=0;**

**for(int i=0;i<MAX;i++)**

**arr[i]=0;**

**}**

**template <class T>**

**void Heap<T>::insert()**

**{**

**int i;**

**cout<<"\n Enter the length of list :";**

**cin>>n;**

**cout<<"\n Enter list elments :";**

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

**cin>>arr[i];**

**}**

**template <class T>**

**void Heap<T>::makeheap()**

**{**

**for(int i=1;i<n;i++)**

**{**

**T val=arr[i];**

**int j=i;**

**int f=(j-1)/2;**

**while(j>0&&arr[f]<val)**

**{**

**arr[j]=arr[f];**

**j=f;**

**f=(j-1)/2;**

**}**

**arr[j]=val;**

**}**

**}**

**template <class T>**

**void Heap<T>::heapsort()**

**{**

**for(int i=n-1;i>0;i--)**

**{**

**T temp=arr[i];**

**arr[i]=arr[0];**

**int k=0;**

**int j;**

**if(i==1)**

**j=-1;**

**else**

**j=1;**

**if(i>2&&arr[2]>arr[1])**

**j=2;**

**while(j>=0&& temp <arr[j])**

**{**

**arr[k]=arr[j];**

**k=j;**

**j=2\*k+1;**

**if(j+1<=i-1&&arr[j]<arr[j+1])**

**j++;**

**if(j>i-1)**

**j=-1;**

**}**

**arr[k]=temp;**

**}**

**}**

**template <class T>**

**void Heap<T>::display()**

**{**

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

**cout<<arr[i]<<" ";**

**cout<<"\n";**

**}**

**void main()**

**{**

**Heap <int> Iobj;**

**int n;**

**clrscr();**

**cout<<"\n Heap Sort \n";**

**Iobj.insert();**

**cout<<"\n The Elements are ..."<<endl;**

**Iobj.display();**

**Iobj.makeheap();**

**cout<<"\n Heapefied"<<endl;**

**Iobj.display();**

**Iobj.heapsort();**

**cout<<"\nElements sorted by Heap sort... "<<endl;**

**Iobj.display();**

**getch();**

**}**

**OUTPUT**

**Heap Sort**

**Enter the length of list:**

**5**

**Enter list elements:**

**14**

**9**

**30**

**3**

**11**

**The elements are ...**

**14 9 30 3 11**

**Heapified**

**30 11 14 3 9**

**Elements Sorted after heap sort ...**

**3 9 11 14 30**

**/\*8.C++ Program for Implementation of various operations of B-tree\*/**

**#include<iostream.h>**

**#include <stdio.h>**

**#include <string.h>**

**#include<conio.h>**

**#include <stdlib.h>**

**#define MAX 4**

**#define MIN 2**

**typedef char Type[10];**

**typedef struct Btree**

**{**

**Type key;**

**} BT;**

**typedef struct treenode**

**{**

**int count;**

**BT entry[MAX+1];**

**treenode \*branch[MAX+1];**

**}node;**

**class B**

**{**

**private:**

**node \*root;**

**public:**

**int LT(char \*,char \*);**

**int EQ(char \*,char \*);**

**node \*Search(Type target,node \*root,int \*targetpos);**

**int SearchNode(Type target,node \*current,int \*pos);**

**node \*Insert(BT New,node \*root);**

**int MoveDown(BT New,node \*current,BT \*med,node \*\*medright);**

**void InsertIn(BT med,node \*medright,node \*current,int pos);**

**void Split(BT med,node \*medright,node \*current,int pos,BT \*newmedian, node \*\*newright);**

**void Delete(Type target, node \*\*root);**

**void Del\_node(Type target, node \*current);**

**void Remove(node \*current, int pos);**

**void Successor(node \*current, int pos);**

**void Adjust(node \*current, int pos);**

**void MoveRight(node \*current, int pos);**

**void MoveLeft(node \*current, int pos);**

**void Combine(node \*current, int pos);**

**void InOrder(node \*root);**

**};**

**int B::LT(char \*a,char \*b)**

**{**

**if((strcmp(a,b)) < (0))**

**return 1;**

**else**

**return 0;**

**}**

**int B::EQ(char \*a,char \*b)**

**{**

**if((strcmp(a,b)) == (0))**

**return 1;**

**else**

**return 0;**

**}**

**node\* B::Search(Type target, node \*root, int \*targetpos)**

**{**

**if (root==NULL)**

**return NULL;**

**else if (SearchNode(target, root, targetpos))**

**return root;**

**else**

**return Search(target, root->branch[\*targetpos], targetpos);**

**}**

**int B::SearchNode(Type target,node \*current, int \*pos)**

**{**

**if (LT(target, current->entry[1].key))**

**{**

**\*pos = 0;**

**return 0;**

**}**

**else**

**{**

**for(\*pos = current->count;**

**LT(target, current->entry[\*pos].key) && \*pos > 1; (\*pos)--);**

**return EQ(target, current->entry[\*pos].key);**

**}**

**}**

**node \*B::Insert(BT newentry,node \*root)**

**{**

**BT medentry;**

**node \*medright;**

**node \*New;**

**if (MoveDown(newentry, root, &medentry, &medright))**

**{**

**New = new node;**

**New->count = 1;**

**New->entry[1] = medentry;**

**New->branch[0] = root;**

**New->branch[1] = medright;**

**return New;**

**}**

**return root;**

**}**

**int B::MoveDown(BT New,node \*current,BT \*med,node \*\*medright)**

**{**

**int pos;**

**if (current == NULL)**

**{**

**\*med = New;**

**\*medright = NULL;**

**return 1;**

**}**

**else**

**{**

**if(SearchNode(New.key, current, &pos))**

**cout<<"Duplicate key !!";**

**if(MoveDown(New, current->branch[pos], med, medright))**

**if (current->count < MAX)**

**{**

**InsertIn(\*med, \*medright, current, pos);**

**return 0;**

**}**

**else**

**{**

**Split(\*med, \*medright, current, pos, med, medright);**

**return 1;**

**}**

**return 0;**

**}**

**}**

**void B::InsertIn(BT med,node \*medright,node \*current, int pos)**

**{**

**int i;**

**for (i = current->count; i > pos; i--)**

**{**

**current->entry[i+1] = current->entry[i];**

**current->branch[i+1] = current->branch[i];**

**}**

**current->entry[pos+1] = med;**

**current->branch[pos+1] = medright;**

**current->count++;**

**}**

**void B::Split(BT med,node \*medright,node \*current, int pos, BT \*newmedian,node \*\*newright)**

**{**

**int i;**

**int median;**

**if (pos <= MIN)**

**median = MIN;**

**else**

**median = MIN + 1;**

**\*newright = new node;**

**for (i = median+1; i <= MAX; i++)**

**{**

**(\*newright)->entry[i - median] = current->entry[i];**

**(\*newright)->branch[i - median] = current->branch[i];**

**}**

**(\*newright)->count = MAX - median;**

**current->count = median;**

**if (pos <= MIN)**

**InsertIn(med, medright, current, pos);**

**else**

**InsertIn(med, medright, \*newright, pos - median);**

**\*newmedian = current->entry[current->count];**

**(\*newright)->branch[0] = current->branch[current->count];**

**current->count--;**

**}**

**void B::Delete(Type target, node \*\*root)**

**{**

**node \*Prev;**

**Del\_node(target,\*root);**

**if ((\*root)->count == 0)//empty root**

**{**

**Prev = \*root;**

**\*root = (\*root)->branch[0];**

**free(Prev);**

**}**

**}**

**void B::Del\_node(Type target,node \*current)**

**{**

**int pos;**

**if (!current)**

**{**

**cout<<"Item not in the B-tree.";**

**return;**

**}**

**else**

**{**

**if (SearchNode(target, current, &pos))**

**if (current->branch[pos-1])**

**{**

**Successor(current, pos);**

**Del\_node(current->entry[pos].key,current->branch[pos]);**

**}**

**else**

**Remove(current, pos);**

**else**

**Del\_node(target, current->branch[pos]);**

**if (current->branch[pos])**

**if (current->branch[pos]->count < MIN)**

**Adjust(current, pos);**

**}**

**}**

**void B::Remove(node \*current, int pos)**

**{**

**int i;**

**for (i = pos+1; i <= current->count; i++)**

**{**

**current->entry[i-1] = current->entry[i];**

**current->branch[i-1] = current->branch[i];**

**}**

**current->count--;**

**}**

**void B::Successor(node \*current, int pos)**

**{**

**node \*leaf;**

**for (leaf=current->branch[pos];leaf->branch[0];**

**leaf = leaf->branch[0]);**

**current->entry[pos] = leaf->entry[1];**

**}**

**void B::Adjust(node \*current, int pos)**

**{**

**if (pos == 0)**

**if (current->branch[1]->count > MIN)**

**MoveLeft(current, 1);**

**else**

**Combine(current, 1);**

**else if (pos == current->count)**

**if (current->branch[pos-1]->count > MIN)**

**MoveRight(current, pos);**

**else**

**Combine(current, pos);**

**else if (current->branch[pos-1]->count > MIN)**

**MoveRight(current, pos);**

**else if (current->branch[pos+1]->count > MIN)**

**MoveLeft(current, pos+1);**

**else**

**Combine(current, pos);**

**}**

**void B::MoveRight(node \*current, int pos)**

**{**

**int i;**

**node \*t;**

**t = current->branch[pos];**

**for (i = t->count;i > 0;i--)**

**{**

**t->entry[i+1] = t->entry[i];**

**t->branch[i+1] = t->branch[i];**

**}**

**t->branch[1] = t->branch[0];**

**t->count++;**

**t->entry[1] = current->entry[pos];**

**t = current->branch[pos-1];**

**current->entry[pos] = t->entry[t->count];**

**current->branch[pos]->branch[0] = t->branch[t->count];**

**t->count--;**

**}**

**void B::MoveLeft(node \*current, int pos)**

**{**

**int c;**

**node \*t;**

**t = current->branch[pos-1];**

**t->count++;**

**t->entry[t->count] = current->entry[pos];**

**t->branch[t->count] = current->branch[pos]->branch[0];**

**t = current->branch[pos];**

**current->entry[pos] = t->entry[1];**

**t->branch[0] = t->branch[1];**

**t->count--;**

**for (c = 1; c <= t->count; c++)**

**{**

**t->entry[c] = t->entry[c+1];**

**t->branch[c] = t->branch[c+1];**

**}**

**}**

**void B::Combine(node \*current, int pos)**

**{**

**int c;**

**node \*right;**

**node \*left;**

**right = current->branch[pos];**

**left = current->branch[pos-1];**

**left->count++;**

**left->entry[left->count] = current->entry[pos];**

**left->branch[left->count] = right->branch[0];**

**for (c = 1; c <= right->count; c++)**

**{**

**left->count++;**

**left->entry[left->count] = right->entry[c];**

**left->branch[left->count] = right->branch[c];**

**}**

**for (c = pos; c < current->count; c++)**

**{**

**current->entry[c] = current->entry[c+1];**

**current->branch[c] = current->branch[c+1];**

**}**

**current->count--;**

**free(right);**

**}**

**void B::InOrder(node \*root)**

**{**

**int pos;**

**if (root)**

**{**

**InOrder(root->branch[0]);**

**for (pos = 1; pos <= root->count; pos++)**

**{**

**cout<<" "<<root->entry[pos].key;**

**InOrder(root->branch[pos]);**

**}**

**}**

**}**

**void main()**

**{**

**int choice,targetpos;**

**Type inKey;**

**BT New;**

**B obj;**

**node \*root, \*target;**

**root = NULL;**

**cout<<"\n\t\t Implementation of B-tree";**

**while(1)**

**{**

**cout<<"\n 1.Insert \n 2.Delete \n 3.Search \n 4.Display";**

**cout<<"\n Enter Your choice";**

**cin>>choice;**

**switch(choice)**

**{**

**case 1:cout<<"Enter the Key to be inserted :";**

**flushall();**

**gets(New.key);**

**root = obj.Insert(New, root);**

**break;**

**case 2:cout<<"Enter the Key to be deleted :";**

**flushall();**

**gets(New.key);**

**cout<<"\n Deleting the desired item..."<<endl;**

**obj.Delete(New.key, &root);**

**break;**

**case 3:cout<<"Enter the Key to be searched for :";**

**flushall();**

**gets(New.key);**

**target = obj.Search(New.key, root, &targetpos);**

**if (target)**

**cout<<"The Searched Item: "<<target->entry[targetpos].key<<endl;**

**else**

**printf("Item is not present\n");**

**break;**

**case 4:cout<<"\n\nInOrder Traversal :\n";**

**obj.InOrder(root);**

**break;**

**default:exit(0);**

**}**

**}**

**}**

**OUTPUT**

**Implementation of B-tree**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:1**

**Enter the Key to be inserted :13**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:1**

**Enter the Key to be inserted :21**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:1**

**Enter the Key to be inserted :34**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:1**

**Enter the Key to be inserted :43**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:1**

**Enter the Key to be inserted :5**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:4**

**InOrder Traversal :**

**5 13 21 34 43**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:2**

**Enter the Key to be deleted :34**

**Deleting the desired item...**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:4**

**InOrder Traversal :**

**5 13 21 43**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:3**

**Enter the Key to be searched for :5**

**The Searched Item: 5**

**1.Insert**

**2.Delete**

**3.Search**

**4.Display**

**Enter Your choice:5**

**/\*9.C++ Program to implement AVL tree\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#include<stdio.h>**

**#define NULL 0**

**struct Node**

**{**

**int n;**

**int bf;**

**struct Node\* left;**

**struct Node\* right;**

**};**

**typedef struct Node node;**

**class AVL**

**{**

**node\* root;**

**public:**

**AVL()**

**{**

**root=NULL;**

**}**

**node\* insert(int data,int\* current)**

**{**

**root=create(root,data,current);**

**return root;**

**}**

**node\* create(node\* root,int data,int\* current);**

**node\* remove(node\* root,int data,int\* current);**

**node\* find\_succ(node\* temp,node \*root,int\* current);**

**node\* right\_rot(node\* root,int\* current);**

**node\* left\_rot(node \*root,int \*current);**

**void display(node \*root);**

**};**

**node\* AVL::create(node \*root,int data,int \*current)**

**{**

**node \*temp1,\*temp2;**

**if(root==NULL)**

**{**

**root=new node;**

**root->n=data;**

**root->left=NULL;**

**root->right=NULL;**

**root->bf=0;**

**\*current=1;**

**return root;**

**}**

**if(data<root->n)**

**{**

**root->left=create(root->left,data,current);**

**if(\*current)**

**{**

**switch(root->bf)**

**{**

**case 1:temp1=root->left;**

**if(temp1->bf==1)**

**{**

**cout<<"\n single LL rotation \n";**

**root->left=temp1->right;**

**temp1->right=root;**

**root->bf=0;**

**root=temp1;**

**}**

**else**

**{**

**cout<<"\n double LR rotation \n";**

**temp2=temp1->right;**

**temp1->right=temp2->left;**

**temp2->left=temp1;**

**root->left=temp2->right;**

**temp2->right=root;**

**if(temp2->bf==1)**

**root->bf=-1;**

**else**

**root->bf=0;**

**if(temp2->bf==-1)**

**temp1->bf=1;**

**else**

**temp1->bf=0;**

**root=temp2;**

**}**

**root->bf=0;**

**\*current=0;**

**break;**

**case 0: root->bf=1;**

**break;**

**case -1: root->bf=0;**

**\*current=0;**

**}**

**}**

**}**

**if(data>root->n)**

**{**

**root->right=create(root->right,data,current);**

**if(\*current)**

**{**

**switch(root->bf)**

**{**

**case 1:root->bf=0;**

**\*current=0;**

**break;**

**case 0: root->bf=-1;**

**break;**

**case -1:temp1=root->right;**

**if(temp1->bf==-1)**

**{**

**cout<<"\n single RR rotation \n";**

**root->right=temp1->left;**

**temp1->left=root;**

**root->bf=0;**

**root=temp1;**

**}**

**else**

**{**

**cout<<"\n double RL rotation\n";**

**temp2=temp1->left;**

**temp1->left=temp2->right;**

**temp2->right=temp1;**

**root->right=temp2->left;**

**temp2->left=root;**

**if(temp2->bf==-1)**

**root->bf=1;**

**else**

**root->bf=0;**

**if(temp2->bf==1)**

**temp1->bf=-1;**

**else**

**temp1->bf=0;**

**root=temp2;**

**}**

**root->bf=0;**

**\*current=0;**

**}**

**}**

**}**

**return root;**

**}**

**void AVL::display(node \*temp)**

**{**

**if(temp!=NULL)**

**{**

**display(temp->left);**

**cout<<temp->n<<" ";**

**display(temp->right);**

**}**

**}**

**node\* AVL::remove(node\* root,int data,int \*current)**

**{**

**node \*temp;**

**if(root==NULL)**

**{**

**cout<<"unavailable";**

**return root;**

**}**

**else**

**{**

**if(data<root->n)**

**{**

**root->left=remove(root->left,data,current);**

**if(\*current)**

**root=right\_rot(root,current);**

**}**

**else**

**{**

**if(data>root->n)**

**{**

**root->right=remove(root->right,data,current);**

**if(\*current)**

**root=left\_rot(root,current);**

**}**

**else**

**{**

**temp=root;**

**if(temp->right==NULL)**

**{**

**root=temp->left;**

**\*current=1;**

**delete(temp);**

**}**

**else**

**{**

**if(temp->left==NULL)**

**{**

**root=temp->right;**

**\*current=1;**

**delete(temp);**

**}**

**else**

**{**

**temp->right=find\_succ(temp->right,temp,current);**

**if(\*current)**

**root=left\_rot(root,current);**

**}**

**}**

**}**

**}**

**}**

**return root;**

**}**

**node\* AVL::find\_succ(node \*succ,node\* temp,int\* current)**

**{**

**node\* temp1=succ;**

**if(succ->left!=NULL)**

**{**

**succ->left=find\_succ(succ->left,temp,current);**

**if(\*current)**

**succ=right\_rot(succ,current);**

**}**

**else**

**{**

**temp1=succ;**

**temp->n=succ->n;**

**succ=succ->right;**

**delete temp1;**

**\*current=1;**

**}**

**return succ;**

**}**

**node\* AVL::right\_rot(node\* root,int\* current)**

**{**

**node \*temp1,\*temp2;**

**switch(root->bf)**

**{**

**case 1: root->bf=0;**

**break;**

**case 0: root->bf=-1;**

**\*current=0;**

**break;**

**case -1:temp1=root->right;**

**if(temp1->bf<=0)**

**{**

**cout<<"\nsingle RR rotation\n";**

**root->right=temp1->left;**

**temp1->left=root;**

**if(temp1->bf==0)**

**{**

**root->bf=-1;**

**temp1->bf=1;**

**\*current=0;**

**}**

**else**

**{**

**root->bf=temp1->bf=0;**

**}**

**root=temp1;**

**}**

**else**

**{**

**cout<<"\ndouble RL rotation\n";**

**temp2=temp1->left;**

**temp1->left=temp2->right;**

**temp2->right=temp1;**

**root->right=temp2->left;**

**temp2->left=root;**

**if(temp2->bf==-1)**

**root->bf=1;**

**else**

**root->bf=0;**

**if(temp2->bf==1)**

**temp1->bf=-1;**

**else**

**temp1->bf=0;**

**}**

**}**

**return root;**

**}**

**node\* AVL::left\_rot(node\* root,int\* current)**

**{**

**node \*temp1,\*temp2;**

**switch(root->bf)**

**{**

**case -1: root->bf=0;**

**break;**

**case 0: root->bf=1;**

**\*current=0;**

**break;**

**case 1: temp1=root->left;**

**if(temp1->bf>=0)**

**{**

**cout<<"\n single LL rotation\n";**

**root->left=temp1->right;**

**temp1->right=root;**

**if(temp1->bf==0)**

**{**

**root->bf=1;**

**temp1->bf=-1;**

**\*current=0;**

**}**

**else**

**{**

**root->bf=temp1->bf=0;**

**}**

**root=temp1;**

**}**

**else**

**{**

**cout<<"\nsingle LR rotation\n";**

**temp2=temp1->right;**

**temp1->right=temp2->left;**

**temp2->left=temp1;**

**root->left=temp2->right;**

**temp2->right=root;**

**if(temp2->bf==1)**

**root->bf=-1;**

**else**

**root->bf=0;**

**if(temp2->bf==-1)**

**temp1->bf=1;**

**else**

**temp1->bf=0;**

**root=temp2;**

**temp2->bf=0;**

**}**

**}**

**return root;**

**}**

**void main()**

**{**

**AVL obj;**

**node \*root=NULL;**

**char ans='n';**

**int current,x,ch;**

**clrscr();**

**do**

**{**

**cout<<"\nEnter choice 1.insert 2.remove 3.display 4.quit";**

**cin>>ch;**

**switch(ch)**

**{**

**case 1: do**

**{**

**cout<<"\nEnter element to be inserted";**

**cin>>x;**

**root=obj.insert(x,&current);**

**cout<<"\ndo u want to enter more elements?(y/n)"<<endl;**

**cin>>ans;**

**}while(ans=='y');**

**break;**

**case 2: cout<<"\nEnter element u want to remove";**

**cin>>x;**

**root=obj.remove(root,x,&current);**

**break;**

**case 3:obj.display(root);**

**break;**

**default:exit(0);**

**}**

**}while(1);**

**}**

**OUTPUT**

**Enter choice 1.insert 2.remove 3.display 4.quit:1**

**Enter element to be inserted:40**

**do u want to enter more elements?(y/n):y**

**Enter element to be inserted:50**

**do u want to enter more elements?(y/n):y**

**Enter element to be inserted:70**

**do u want to enter more elements?(y/n):n**

**Enter choice 1.insert 2.remove 3.display 4.quit:3**

**single RR rotation**

**40 50 70**

**Enter choice 1.insert 2.remove 3.display 4.quit:2**

**Enter element u want to remove:70**

**Enter choice 1.insert 2.remove 3.display 4.quit:3**

**40 50**

**Enter choice 1.insert 2.remove 3.display 4.quit:4**

**/\*10.C++ Program to implement dictionary(ADT) operations using hashing\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#define MAX 10**

**class HashTab**

**{**

**private:**

**struct DCT**

**{**

**int k;**

**int val;**

**}a[MAX];**

**public:**

**int hash(int);**

**void init();**

**void insert(int,int,int);**

**void display();**

**void size();**

**void search(int);**

**};**

**void HashTab::init()**

**{**

**for(int i=0;i<MAX;i++)**

**{**

**a[i].k= -1;**

**a[i].val=-1;**

**}**

**}**

**int HashTab::hash(int num)**

**{**

**int Hkey;**

**Hkey=num%10;**

**return Hkey;**

**}**

**void HashTab::insert(int index,int key,int value)**

**{**

**int flag,i,count=0;**

**flag=0;**

**if(a[index].k==-1)**

**{**

**a[index].k=key;**

**a[index].val=value;**

**}**

**else**

**{**

**i=0;**

**while(i<MAX)**

**{**

**if(a[i].k!= -1)**

**count++;**

**i++;**

**}**

**if(count==MAX)**

**{**

**cout<<"\nHash Table Is Full";**

**}**

**for(i=index+1;i<MAX;i++)**

**if(a[i].k== -1)**

**{**

**a[i].k=key;**

**a[i].val=value;**

**flag=1;**

**break;**

**}**

**for(i=0;i<index&&flag==0;i++)**

**if(a[i].k== -1)**

**{**

**a[i].k=key;**

**a[i].val=value;**

**flag=1;**

**break;**

**}**

**}**

**}**

**void HashTab::display()**

**{**

**int i;**

**cout<<"\n The Hash Table is...\n";**

**for(i=0;i<MAX;i++)**

**{**

**cout<<"\n "<<i<<" "<<a[i].k<<" "<<a[i].val;**

**}**

**}**

**void HashTab::size()**

**{**

**int len=0,i;**

**for(i=0;i<MAX;i++)**

**{**

**if(a[i].k!=-1)**

**len++;**

**}**

**cout<<"\n The size of dictionary is ";**

**cout<<len;**

**}**

**void HashTab::search(int search\_key)**

**{**

**int i,j;**

**i=hash(search\_key);**

**if(a[i].k==search\_key)**

**{**

**cout<<"\n The Record is present at location "<<i;**

**return;**

**}**

**if(a[i].k!=search\_key)**

**{**

**for(j=i;j<MAX;j++)**

**{**

**if(a[j].k==search\_key)**

**{**

**cout<<"\n The Record is present at location "<<j;**

**return;**

**}**

**}**

**for(j=0;j<i;j++)**

**{**

**if(a[j].k==search\_key)**

**{**

**cout<<"\n The Record is present at location "<<j;**

**return;**

**}**

**}**

**}**

**else**

**cout<<"\n The Record is not present in the hash table";**

**}**

**void main()**

**{**

**int key,value,Hkey,search\_key;**

**char ans;**

**HashTab obj;**

**clrscr();**

**cout<<"\nDictionary Functions using Hashing";**

**obj.init();**

**do**

**{**

**cout<<"\n Enter The key";**

**cin>>key;**

**cout<<"\n Enter The Value";**

**cin>>value;**

**Hkey=obj.hash(key);**

**obj.insert(Hkey,key,value);**

**cout<<"\n Do U Wish To Continue?(y/n)";**

**ans=getche();**

**}while(ans=='y');**

**obj.display();**

**obj.size();**

**cout<<"\n Enter the key for searching the record";**

**cin>>search\_key;**

**obj.search(search\_key);**

**getch();**

**}**

**OUTPUT**

**Dictionary functions using hashing**

**Enter the key:14**

**Enter the value:5**

**Do you wish to continue?(y/n)y**

**Enter the key:9**

**Enter the value:11**

**Do you wish to continue?(y/n)y**

**Enter the key:19**

**Enter the value:3**

**Do you wish to continue?(y/n)y**

**Enter the key:30**

**Enter the value:12**

**Do you wish to continue?(y/n)n**

**Hash table**

**0 19 3**

**1 30 12**

**2 -1 -1**

**3 -1 -1**

**4 14 5**

**5 -1 -1**

**6 -1 -1**

**7 -1 -1**

**8 -1 -1**

**9 9 11**

**The size of dictionary is 4**

**Enter the key for searching the record:14**

**The Record is present at location:4**

**/\*11.C++ Program for implementing Knuth-Morris-Pratt pattern matching algorithm\*/**

**#include<iostream.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**class KMP**

**{**

**char \*text,\*pattern;**

**public:**

**KMP()**

**{**

**\*text=NULL;**

**\*pattern=NULL;**

**}**

**void get\_input();**

**void display();**

**void kmp(const char \*text,const char \*pattern);**

**};**

**void KMP::kmp(const char \*text,const char \*pattern)**

**{**

**int i,j,T[50];**

**T[0]=-1;**

**for(i=0;pattern[i]!='\0';i++)**

**{**

**T[i+1]=T[i]+1;**

**while(T[i+1]>0 && pattern[i]!=pattern[T[i+1]-1])**

**T[i+1]=T[T[i+1]-1]+1;**

**}**

**for(i=j=0;text[i]!='\0';)**

**{**

**if(j<0||text[i]==pattern[j])**

**{**

**++i,++j;**

**if(pattern[j]=='\0')**

**{**

**break;**

**}**

**}**

**else**

**j=T[j];**

**}**

**if((i-j+1)>=strlen(text))**

**cout<<"Not found";**

**else**

**cout<<"Pattern matched at "<<i-j+1;**

**}**

**void KMP::get\_input()**

**{**

**cout<<"Enter text \n";**

**cin>>text;**

**cout<<"Enter pattern \n";**

**cin>>pattern;**

**}**

**void KMP::display()**

**{**

**kmp(text,pattern);**

**}**

**void main()**

**{**

**KMP obj;**

**clrscr();**

**obj.get\_input();**

**obj.display();**

**getch();**

**}**

**OUTPUT:**

**Enter text:**

**fantastic**

**Enter pattern:**

**ant**

**Pattern matched at 2**

**/\*12.C++ Program to implement Boyer Moore Pattern matching algorithm.\*/**

**#include<string.h>**

**#include<iostream.h>**

**#include<conio.h>**

**#define NO\_OF\_CHARS 256**

**int max (int a, int b)**

**{**

**return (a > b)? a: b;**

**}**

**void badCharHeuristic( char \*str, int size, int badchar[NO\_OF\_CHARS])**

**{**

**int i;**

**for (i = 0; i < NO\_OF\_CHARS; i++)**

**badchar[i] = -1;**

**for (i = 0; i < size; i++)**

**badchar[(int) str[i]] = i;**

**}**

**void search( char \*txt, char \*pat)**

**{**

**int m = strlen(pat);**

**int n = strlen(txt);**

**int flag=0;**

**int badchar[NO\_OF\_CHARS];**

**badCharHeuristic(pat, m, badchar);**

**int s = 0;**

**while(s <= (n - m))**

**{**

**int j = m-1;**

**while(j >= 0 && pat[j] == txt[s+j])**

**j--;**

**if (j < 0)**

**{**

**cout<<"\nPattern occurs at "<<s;**

**flag++;**

**s += (s+m < n)? m-badchar[txt[s+m]] : 1;**

**}**

**else**

**s += max(1, j - badchar[txt[s+j]]);**

**}**

**if(flag==0)**

**cout<<"\nPattern NOT present !!!";**

**}**

**int main()**

**{**

**char text[40],pattern[20];**

**clrscr();**

**cout<<"\nBoyer-Moore-Algorithm\n";**

**cout<<"\nEnter text string :";**

**cin>>text;**

**cout<<"\nEnter pattern string : ";**

**cin>>pattern;**

**search(text,pattern);**

**getch();**

**return 0;**

**}**

**OUTPUT :**

**Boyer-Moore-Algorithm\n";**

**Enter text string : abbbacabccabc**

**Enter pattern string : abc**

**Pattern occurs at 6**

**Pattern occurs at 10**