#include <iostream>  
#include <string>  
  
using namespace std;  
#define size 10  
  
class Hash  
{  
 private:  
    struct tel  
    {  
        string emp\_name;  
        string comp\_name;  
        int id;  
    }tel[size];  
     
    public:  
        void insert();  
        void display();  
        void collision\_lp(int,string,string,int);  
        void collision\_qp(int,string,string,int);  
        //void hash\_fun(long int);  
         
        //constructor to initialize values  
        Hash()  
        {  
            for(int i=0;i<size;i++)  
            {  
                tel[i].id=0;  
                tel[i].emp\_name="--";  
                tel[i].comp\_name="--";  
            }  
        }  
};  
  
void Hash::insert()  
{  
    int choice;  
    string name;  
    string cname;  
    int id\_no;  
    int loc;  
    cout<<"Enter employee id: ";  
    cin>>id\_no;  
    cout<<"\nEnter name: ";  
    cin>>name;  
    cout<<"Enter company name: ";  
    cin>>cname;  
     
    loc=id\_no%size;  
     
    //if(tel[loc].id==0)  
     
        if(tel[loc].id==0)  
        {  
       tel[loc].emp\_name=name;  
       tel[loc].comp\_name=cname;  
       tel[loc].id=id\_no;  
       display();  
        }  
     
     else  
    {  
        cout<<"\nCollision occured\n";  
        cout<<"\n1.Linear probing\n2.Quadratic probing: ";  
        cin>>choice;  
        switch(choice)  
        {  
          case 1:  
            collision\_lp(loc,name,cname,id\_no);  
            cout<<"\n";  
            display();  
            break;  
          case 2:  
            collision\_qp(loc,name,cname,id\_no);  
            cout<<"\n";  
            display();  
            break;  
        }  
    }  
}  
void Hash::collision\_lp(int Loc,string Name,string Cname,int id\_No)  
{  
int k,new\_loc;  
for(k=0;k<size;k++)  
{  
new\_loc=(Loc+k)%size;  
if(tel[new\_loc].id==0)  
        {  
          tel[new\_loc].emp\_name=Name;  
          tel[new\_loc].comp\_name=Cname;  
          tel[new\_loc].id=id\_No;  
          break;  
        }  
  
}  
  
  
}  
  
void Hash::collision\_qp(int Loc,string Name,string Cname,int id\_No)  
{  
int k,new\_loc;  
for(k=0;k<size;k++)  
{  
new\_loc=(Loc+(k\*k))%size;  
if(tel[new\_loc].id==0)  
        {  
          tel[new\_loc].emp\_name=Name;  
          tel[new\_loc].comp\_name=Cname;  
          tel[new\_loc].id=id\_No;  
          break;  
        }  
  
}  
  
  
}  
  
void Hash::display()  
{  
int i;  
cout<<"Employee id\t\tEmployee name\t\tCompany name"<<endl;  
for(i=0;i<size;i++)  
{  
    cout<<tel[i].id <<"\t\t\t" <<tel[i].emp\_name<<"\t\t\t"<<tel[i].comp\_name<<endl;  
  
}  
  
  
}  
  
int main()  
{  
    int j,elements;  
    cout<<"Enter no of elements: ";  
    cin>>elements;  
    Hash obj;  
    for(j=0;j<elements;j++)  
    {  
      obj.insert();  
    }  
    cout<<"\nFinal hash table is:\n";  
    obj.display();  
     
     
  
    return 0;  
}

#include<iostream>

using namespace std;

class node

{

public:

int data;

node \* next;

};

class list

{

node \*head,\*temp;

public:

void create();

node\* getnode();

//node \*newnode();

void display();

void Insert\_at\_Pos( node \*newnode, int position);

void append(node \*newnode);

void traversal();

void reverse();

void search();

list()

{

head=NULL;

}

};

// create function

void list:: create()

{

node \* newnode;

char choice;

while(1){

cout<<"\nDo you want to create new node (Y/N) :";

cin>>choice;

if(choice=='N')

break;

newnode = getnode();

append(newnode);

}

}

/\*//append

void list :: append()

{

node \*temp;

if(head==NULL){

head=newnode;

temp=newnode;

}

else{

temp->next=newnode;

temp=newnode;

}

}\*/

//getnode functin

node\* list::getnode()

{

node \* newnode;

newnode = new node;

cout<<"\nEnter data :";

cin>>newnode->data;

newnode->next=NULL;

return (newnode);

}

//display function

void list::display()

{

//node \* temp;

temp=head;

if(temp==NULL){

cout<<"The list is empty"<<endl;

}

else{

while(temp != NULL){

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

temp=temp->next;

}

}

}

//insertion

void list :: Insert\_at\_Pos(node \*newnode, int position)

{

node \*temp = head;

int count=1, flag=1;

if(position==1)

{

newnode->next=temp;

head = newnode;

}

else

{

while(count != position-1)

{

temp = temp->next;

if(temp==NULL)

{

flag=0;

break;

}

count++;

}

if(flag==1)

{

newnode->next = temp->next;

temp->next = newnode;

}

else

{

cout<<"Position not found";

}

}

}

//append

void list :: append(node\* newnode)

{

//node \*temp=head;

if(head == NULL)

{

head = newnode;

temp=head;

}

else

{

temp->next = newnode;

temp = temp->next;

}

}

//traverse

void list :: traversal()

{

node \*temp = head;

if(temp==NULL)

{

cout<<"List is empty";

}

else

{

while(temp!=NULL)

{

cout<<temp->data;

temp=temp->next;

}

}

}

//reverse

void list :: reverse()

{

node \*prev=NULL;

node \*nextt=NULL;

node \*temp=head;

while(temp!=NULL)

{

nextt==temp->next;

temp->next=prev;

prev=temp;

temp=nextt;

}

head=prev;

}

//search

void list :: search()

{

int val;

cout<<"Enter the value to be searched:";

cin>>val;

node \*temp=head;

int flag=0;

while(temp!=NULL)

{

if(temp->data==val)

{

cout<<"Value found:";

flag=1;

break;

}

temp=temp->next;

}

if(flag==0)

{

cout<<"Element not found:";

}

}

int main()

{

int position,choice;

node \*newnode;

list L1;

while(1)

{

cout<<"CHOOSE WHAT YOU WANT TO PERFORM:\n1)create\n2)display\n3)get\_node\n4)insert\_at\_position\n5)delete\n6)reverse\n7)search\n8)exit"<<endl;

cin>>choice;

switch(choice)

{

case 1:

{

L1.create();

break;

}

case 2:

{

L1.display();

break;

}

case 3:

{

newnode = L1.getnode();

break;

}

case 4:

{

cout<<"enter new position";

cin>>position;

L1.Insert\_at\_Pos(newnode, position);

//L1.display();

break;

}

case 5:

{

L1.traversal();

break;

}

case 6:

{

L1.reverse();

break;

}

case 7:

{

L1.search();

break;

}

// L1.display();

}

}

return 0;

}

//STACK CODE  
  
  
#include <iostream>  
  
using namespace std;  
  
class stack\_node  
{  
    public:  
        int data;  
        stack\_node \*next;  
};  
  
class Stack\_New  
{  
    private:  
        stack\_node \*top;  
        int size;  
     
    public:  
        int is\_empty();  
        int pop();  
        void push(int element);  
        int Gettop();  
     
    Stack\_New()  
    {  
        top = NULL;  
        size=0;  
    }  
};  
  
int Stack\_New :: is\_empty()  
{  
    if(top==NULL)  
     
        return 1;  
     
    else  
     
        return 0;  
     
}  
  
int Stack\_New :: Gettop()  
{  
    if(!is\_empty())  
     
        return(top->data);  
     
}  
  
void Stack\_New :: push(int value)  
{  
    stack\_node \*NewNode;  
    NewNode = new stack\_node;  
    NewNode->data = value;  
    NewNode->next = top;  
    top = NewNode;  
}  
  
int Stack\_New :: pop()  
{  
    int data;  
    stack\_node \*temp = top;  
    data = top->data;  
    if(is\_empty())  
     
        cout<<"Stack Underflow"<<endl;  
     
    else  
     
     {   top = top->next;  
        delete(temp);  
        return data;  
     }  
}  
  
  
int main()  
{  
    Stack\_New s;  
    int choice,element;  
    while(1)  
    {  
        cout<<"Enter the action to be performed\n1.Push\n2.pop\n3.Gettop\n4.Exit"<<endl;  
        cin>>choice;  
        switch(choice)  
        {  
            case 1:  
            {  
               cout<<"Enter element to be added:";  
               cin>>element;  
               s.push(element);  
               break;  
            }  
             
            case 2:  
            {  
                cout<<"Popped element is:"<<s.pop()<<endl;  
                break;  
            }  
             
            case 3:  
            {  
                cout<<"Top element is:"<<s.Gettop()<<endl;  
                break;  
            }  
             
            case 4:  
            {  
                break;  
            }  
        }  
    }  
     
  
    return 0;  
}

//QUEUE CODE  
  
#include <iostream>  
  
using namespace std;  
  
class QueueNode  
{  
    public:  
    int data;  
    QueueNode \*next;  
};  
  
class QueueNew  
{  
    private:  
        QueueNode \*front, \*rear;  
    public:  
        int empty();  
        void insert(int element);  
        int Delete();  
        int Getfront();  
         
    QueueNew()  
    {  
        front = rear = NULL;  
    }  
};  
  
int QueueNew :: empty()  
{  
    if(front = NULL)  
        return 1;  
    else  
        return 0;  
}  
  
void QueueNew :: insert(int value)  
{  
    QueueNode \*NewNode;  
    NewNode = new QueueNode;  
    NewNode->data =  value;  
    NewNode->next = NULL;  
    if(rear == NULL)  
    {  
        front = NewNode;  
        rear = NewNode;  
    }  
    else  
    {  
        rear->next = NewNode;  
        rear = NewNode;  
    }  
}  
  
int QueueNew :: Delete()  
{  
    int temp;  
    QueueNode \*curr = NULL;  
    if(!empty())  
    {  
        temp = front->data;  
        curr = front;  
        front = front->next;  
        delete curr;  
        return temp;  
    }  
    else  
    {  
        cout<<"Queue is empty";  
        return 0;  
         
    }  
}  
  
int QueueNew :: Getfront()  
{  
    if(!empty())  
    {  
        return(front->data);  
    }  
    else  
        return 0;  
}  
  
int main()  
{  
   QueueNew Q;  
   int choice, element;  
    while(1)  
    {  
        cout<<"Enter the action to be performed\n1.INsert\n2.Delete\n3.Getfront\n4.Exit"<<endl;  
        cin>>choice;  
        switch(choice)  
        {  
            case 1:  
            {  
               cout<<"Enter element to be added:";  
               cin>>element;  
               Q.insert(element);  
               break;  
            }  
             
            case 2:  
            {  
                cout<<"DEleted element is:"<<Q.Delete()<<endl;  
                break;  
            }  
             
            case 3:  
            {  
                cout<<"Front element is:"<<Q.Getfront()<<endl;  
                break;  
            }  
             
            case 4:  
            {  
                break;  
            }  
        }  
    }  
     
  /\* Q.insert(11);  
   Q.insert(12);  
   cout<<endl<<Q.Delete()<<endl;  
   cout<<endl<<Q.Getfront()<<endl;\*/  
   return 0;  
}