Phone Directory Application Using Doubly-Linked Lists

1. ABSTRACT

Phone Directory applications is one of the most used applications by everyone.It consists of searching and sorting criteria. We can search by phone number, name and mail id to get the required information from the phonebook. Search results will return one or multiple items and the list appears in a sorted order.We will implement such a system where phone data will be stored in a DoublyLinkedList.

Search results are loaded into a doubly linked list, where items are sorted automatically while displaying. This way our results will not only be automatically sorted, but if there are any duplicates present in the doubly linked list they will not appear in the sorted result.

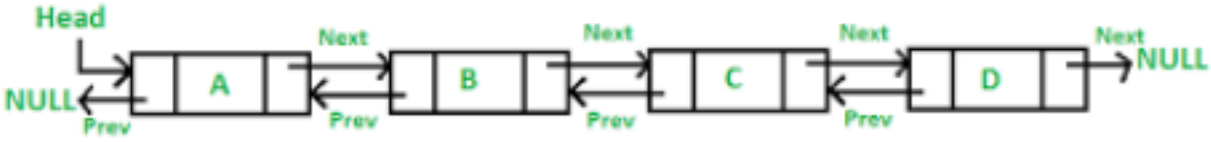
A Person's name, phone number, mail id are to be inserted into the doubly linked list by the user based on their arrival, but then they are displayed based on the alphabetical order of their names.

The Deletion of any entry in the phonebook can be done by entering the name of the person whose data to be deleted , by that the entire information of that person will be eliminated from the list.

Updating can be done on any field (phone number, name, mail id)of the contact and can be updated in respective ways. We update all fields of a contact at a time or a single field at once. All the details of a person entered will be displayed and also the operations performed will appear after getting updated.

2. INTRODUCTION

A doubly-linked list is particularly feasible because you can iterate and search from the front (head) of the list or back (tail) of the list. Doubly linked list(DLL) is a type of linked list in which each node apart from storing its data has two links. The first link points to the previous node in the list and the second link points to the next node in the list. The first node of the list has its previous link pointing to null similarly the last node of the list has its next node pointing to null.



A DLL can be traversed in both forward and backward directions. The delete operation in DLL is more efficient when a pointer to the node to be deleted is given. We can quickly insert a new node before a given node. In singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.

Every node of DLL requires extra space for a previous pointer. It is possible to implement DLL with a single pointer though. All operations require an extra pointer previous to be maintained.

For example, in insertion, we need to modify previous pointers together with next pointers. For example, in the following functions for insertions at different positions, we need 1 or 2 extra steps to set the previous pointer.

A node can be added in four ways
1) At the front of the DLL
2) After a given node.
3) At the end of the DLL
4) Before a given node.

Memory Representation of a doubly linked list

Memory Representation of a doubly linked list is shown in the following image. Generally, a doubly linked list consumes more space for every node and therefore, causes more expansive basic operations such as insertion and deletion. However, we can easily manipulate the elements of the list since the list maintains pointers in both the directions (forward and backward).

In the following image, the first element of the list that is i.e. 13 stored at address 1. The head pointer points to the starting address 1. Since this is the first element being added to the list therefore the previous of the list contains null. The next node of the list resides at address 4 therefore the first node contains 4 in its next pointer.

We can traverse the list in this way until we find any node containing null or -1 in its next part.

3. RELATED WORKS/LITERATURE SURVEY

There were many existing phone book directory applications, performed using various operations like oop in c++, file handling in data structures, single linked lists, hash tables, tires etc.

- One of the already existing applications is the Phone Book application developed in C++ using the OOP principles.The

technique is by activity the implementation of details within a category. This application is developed using a hash table and tire data structure which is quoted as a most suitable operation for a phonebook.[6]

- Another one is a mini project using console applications without graphics implemented using c++ or c. File handling and data structure concepts have been extensively used for almost all functions in this mini project.[7]

- Another is the Phonebook management system project in C++,a simple console application built without graphics. In this project, users can add a new phone record, display existing phone records, search a particular phone record and delete phone records. It consists of file handling operations such as how to add, search, modify, list and delete records using file.[8]
- The application program has been implemented using experimental cases and the language used is Cor C++. This application works for other functions that make it easy to search, delete, edit, and remember our peer information.[9].

- This project developed by basic concepts of the functions, file handling, and data structure.It is made for beginners how to add, list, modify or edit, search, and delete data to/from the file.The basic functions are to add new records, listing them, modifying them and updating, searching for contacts, and deleting the phonebook records. It is developed using the c++ language.[10]

Upon all these existing projects our application is the efficient one as it is implemented using doubly linked lists and performs operations like insertion,deletion,update,search and sorting from both sides forward and backward, which makes people use the application easily.

4. PROBLEM STATEMENT/OBJECTIVE

Our Objective is to make an efficient phone directory application that works with max of O(n) time complexity and min of O(1). we made it user friendly like the code is menu driven with multiple functions such as:

1.Searching:

Searching can be done by 3 ways, you can search by name,number or mail id.
And searching takes O(n) complexity, at the worst case.

2.Sorting:

Sorting will be done automatically when we display the phone directory.The names get sorted according to alphabetical order.

3.Deleting:

Deleting can be done by name of the contact and its time complexity is O(n) at worst case and O(1) is best case complexity.

4.Updating:

We can update any field information and updating takes O(n) at worst case and O(1) at best case.

5.Displaying:

It displays all the contacts in the phone directory and it takes O(nlogn) time complexity as it first sorts and displays as sorting takes O(nlogn) time complexity at worst case.

5. PROPOSED SYSTEM

Now let us see the detailed working of each function in our application.

Firstly **insertion(Pseudo code):**

```
insert()
        num[50];
         gmail[40]
         name[30]
          ans;
           Do
              Enter name;
               while (len(number)!=10)
                       Enter valid number;
              end-while
                Enter G-mail;
                Create new node;
              if(head==NULL)
                   Return head;
               End-if
              Else
                  Traverse through entire linked list
               End-else
             End-do
```

Next we will see about **deletion(Pseudo code):**

```
delete name(name)
        c=0;
        ptr=head;
        while(temp!=NULL)
           if(string is same as name)
                c=1;
              Break; end-if;
            Else
             c=2;
            End-else;
            Traverse; end-while;
            if(c=1 and ptr!=head and ptr->next!=NULL) //middle element
                        Connect the link of previous node with next;
                  delete(ptr->name)
            End-if
            if(ptr==head)
                  delete(ptr->name);
                  Increment head to next;
            End-if
            if(ptr->next==NULL)
                  delete(ptr->name);
                   Make the next pointer as null;
             End-if
            if(c==2)
              Name not present in the linked list;
            End-if;
```

Now in the same way we can delete the number and email by just replacing the ptr->name to ptr-> gmail or ptr-> contact number. Now lets see how to display contact number,name and gmail

Display(Pseudo Code):

```
display()
    ptr=head;
    while(ptr!=NULL)
      Display name;
      Display contact number;
      Display Gmail;
      Traverse;
     end -while;
```

Next let's see about searching

Search by name(Pseudo Code):

```
SearchByName(name)
    ptr=head;
    while(ptr!=NULL)
        if(name matches name which is taken as argument)
            Display “Name is found”;
            Display all the necessary information;
         End-if;
         Traverse through entire list;
         Else
              Display “Name not found”;
       End-while
```

In the same way we can search for gmail and contact number by just changing the function arguments and displaying whether the number of gmail found or not found.

Now finally lets see another powerful operation which is sorting.

Sorting(Pseudo code):

```
sorting()
  Pointer i , pointer j;
```

```
Temp;
 Node;
 for(i=head;i->next!=NULL;i=i->next)
      for(j=i->next;j!=NULL;j=j->next)
           if(node>0)
              Temp = i->name;
               i->name=j->name;
              j->name=Temp;
           End-if;
       End-for;
End-for;
```

Here we can even implement other sorting techniques such as quicksort , mergesort etc… , so this sorting happens as soon as the display is called so we will be calling sort function in display also.

Now finally lets see about the last operation that is updating.

Update(Pseudo code):

```
update(node)
   Ans;
    C;
    ptr=head;
    while(ptr!=NULL)
       if( you find the node you want to update)
          Do
               Update any detail by using menu driven program
               if(len(number)!=10)
                  Enter valid number;
          End-do
       End-if
       Traverse till the last node;
```

End-while;

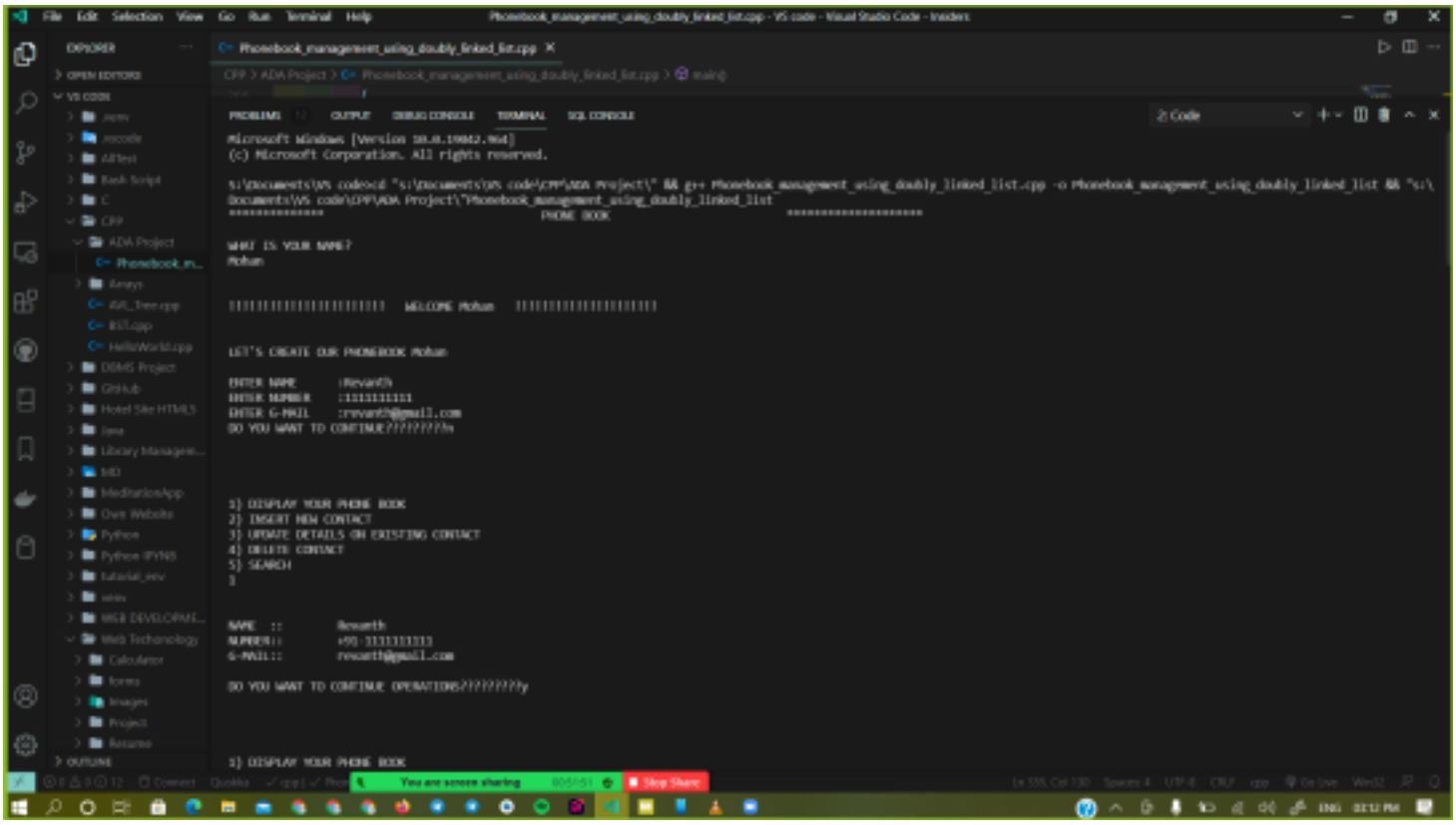
So these are the brief operations of our application.

System Requirements:

- ❖ Windows 7/8/8.1/10
- ❖ Intel i3(minimum)
- ❖ 2GB RAM
- ❖ 1.2 GHz Processor

6. RESULTS/PERFORMANCE ANALYSIS

1. Inserting contacts into phonebook:



2. Updating a contact:

3. Searching for a contact by name, phone number and mail id:

4. Deleting a contact:

5. Displaying phone book with all contacts:

7. CONCLUSION

Phone directory has become a mandatory application for everyone around.One can easily enter,update or delete contact from their phonebook by this application.This application is developed using doubly linked list which makes searching and sorting facile.

We have developed this application in a simplified way, so that any person can access it without much effort. People can add a contact into their phonebook, update a contact's name,phone number or mail id, and delete that contact from their phonebook.

The operations take place with very much ease, we have used insertion, deletion, searching, sorting,displaying techniques of doubly

linked list into this phonebook application ,as DLL is a feasible data structure that can be traversed from both sides i.e, forward and backward, also takes less time for performing operations.

Time complexity:

1. Searching - O(n)
2. Deletion - O(n)
3. Update - O(n)
4. Displaying - O(n logn)

8. REFERENCES

[1] Implement a Phone Directory

[2]https://github.com/PRITI24/Phonebook-management-using-doubly-linked-list

[3]Project 6: Mobile Phone Directory Using Doubly Lin... [4]

Realization of phone book C++ code by doubly linked list

[5]application of doubly linked list

[6]https://www.codingninjas.com/blog/2020/10/15/implementing-a-phon

e-directory-in-c/

[7]https://itsourcecode.com/free-projects/c-projects/phone-book-management-system-in-c-with-source-code/

[8]https://www.codewithc.com/phonebook-management-system-project in-c/

[9]https://1000projects.org/phonebook-application-c-project-report.html

[10]https://projectnotes.org/it-projects/phonebook-system-in-c-c-with-source-code/

9. CODE

```
#include <iostream>
#include<cstring>
using namespace std;
class dnode
{
    public:
        char number[50];
        char gmail[40];
        char name[30];
        dnode *prev,*next;
        dnode(char n[],char r[],char g[])
{
                strcpy(name,n);
                 strcpy(number,r);
                strcpy(gmail,g);
                 next=NULL;
                 prev=NULL;
}
```

```
    friend class dlist;
};
class dlist
{
    dnode *head,*temp,*ptr;

    dnode *ptr1, *ptr2, *dup;
    public:
    dnode *prevn;

    void insert();
    void sort();
    void deletecontact(char n[20]);
    void deletesamenumber();
    void deletesamename();
    void deletesamegmail();
    void searchbyname(char n[20]);
    void searchbynumber(char no[30]);
    void searchbygmail(char g[20]);

        // dnode *head,*temp,*ptr;
        // friend class apply;
        void accept();
        void display();
        void update(char ch[10]);
        dlist()
                {
                        head=NULL;
                        temp=NULL;
                        ptr=NULL;
                        ptr1=NULL;
                        ptr2=NULL;
                        dup=NULL;
                }
};
// class apply()
// {

// }
```

```
void dlist::accept()
{
        char number[50];
        char gmail[40];
        char name[30];
    char ans;
      do
        {
    cout<<"ENTER NAME :";
    cin>>name;
    // cin.getline (name,30);
    cout<<"ENTER NUMBER :";
    cin>>number;
    while(strlen(number)!=10)
    {
    cout<<"ENTER VALID NUMBER :";
    cin>>number;
    }
    cout<<"ENTER G-MAIL :";
    cin>>gmail;
    temp=new dnode(name,number,gmail);
    if(head==NULL)
    {
        head=temp;
    }
    else
    {
       ptr=head;
        while(ptr->next!=NULL)
        {
ptr=ptr->next;
        }
        ptr->next=temp;
        temp->prev=ptr;
    }
        cout<<"DO YOU WANT TO CONTINUE?????????";
    cin>>ans;
}while(ans=='y');
```

```
}
void dlist::display()
{
    ptr=head;//start the node
    while(ptr!=NULL)//traverse till last
    {
            cout<<"\n\tNAME ::\t"<<ptr->name;
            cout<<"\nNUMBER::\t+91-"<<ptr->number;
            cout<<"\nG-MAIL::\t"<<ptr->gmail;
            ptr=ptr->next;
    }
}

    void dlist::insert()
    {
        accept();
    }
    void dlist::sort()
    {
        dnode *i,*j;
        int temp;
        char n[10];
        for(i=head;i->next!=NULL;i=i->next)
        {
            for(j=i->next;j!=NULL;j=j->next)
            {
                temp=strcmp(i->name,j->name);
                if(temp>0)
                {
                   strcpy(n,i->name);
                   strcpy(i->name,j->name);
                   strcpy(j->name,n);
                }
            }
        }

    }
    void dlist::deletecontact(char s[20])
```

```
{
    int c=0;
    ptr=head;
    while(ptr!=NULL)
    {
        if(strcmp(s,ptr->name)==0)
        {
            c=1;
            break;
        }
        else
        {
            c=2;
        }
        ptr=ptr->next;
    }
    if(c==1 && ptr!=head && ptr->next!=NULL)
    {
        ptr->prev->next=ptr->next;
       ptr->next->prev=ptr->prev;
      delete(ptr);
      cout<<"YOUR CONTACT IS SUCCESSFULLY DELETED\n\n";
    }
    if(ptr==head)
    {
        head=head->next;
        head->prev=NULL;
        delete(ptr);
      cout<<"YOUR CONTACT IS SUCCESSFULLY DELETED\n\n";
    }
    if(ptr->next==NULL)
    {
        ptr->prev->next=NULL;
        ptr->prev=NULL;
        delete(ptr);
      cout<<"YOUR CONTACT IS SUCCESSFULLY DELETED\n\n";
    }
    if(c==2)
    {
```

```
        cout<<"YOUR ENTERED NAME IS NOT IN THE LIST...";
    }
}
void dlist::deletesamename()
{
    ptr1=head;
    while (ptr1 != NULL && ptr1->next != NULL)
{
    ptr2 = ptr1;
    while (ptr2->next != NULL)
    {
        if (strcmp(ptr1->name,ptr2->next->name)==0)
        {
            dup = ptr2->next;
            ptr2->next = ptr2->next->next;
            delete(dup);
        }
        else
        {
            ptr2 = ptr2->next;
    }
    }
    ptr1 = ptr1->next;
}
}
void dlist::deletesamegmail()
{
    ptr1=head;
    while (ptr1 != NULL && ptr1->next != NULL)
{
    ptr2 = ptr1;
    while (ptr2->next != NULL)
    {
        if (strcmp(ptr1->gmail,ptr2->next->gmail)==0)
        {
            dup = ptr2->next;
            ptr2->next = ptr2->next->next;
            delete(dup);
        }
```

```
        else
        {
            ptr2 = ptr2->next;
    }
    }
    ptr1 = ptr1->next;
}
}
void dlist::deletesamenumber()
{
 ptr1=head;
    while (ptr1 != NULL && ptr1->next != NULL)
{
    ptr2 = ptr1;
    while (ptr2->next != NULL)
    {
        if (strcmp(ptr1->number,ptr2->number)==0)
        {
            dup = ptr2->next;
            ptr2->next = ptr2->next->next;
            delete(dup);
        }
        else
        {
            ptr2 = ptr2->next;
    }
    }
    ptr1 = ptr1->next;
}
}
void dlist::searchbyname(char na[10])
{
    ptr=head;
    while(ptr!=NULL)
    {
        if(strcmp(na,ptr->name)==0)
        {
            cout<<"NAME FOUND"<<endl;
            cout<<"CONTACT DETAILS ARE BELOW:\n"<<endl;
```

```
            cout<<"\n\tNAME ::\t"<<ptr->name;
            cout<<"\nNUMBER::\t+91-"<<ptr->number;
            cout<<"\nG-MAIL::\t"<<ptr->gmail;

            }
            ptr=ptr->next;
        }
}
void dlist::searchbynumber(char num[20])
{
 ptr=head;
    while(ptr!=NULL)
    {
        if(strcmp(num,ptr->number)==0)
        {
            cout<<"NUMBER FOUND\n"<<endl;
            cout<<"CONTACT DETAILS ARE BELOW:\n"<<endl;
        cout<<"\n\tNAME ::\t"<<ptr->name;
        cout<<"\nNUMBER::\t+91-"<<ptr->number;
        cout<<"\nG-MAIL::\t"<<ptr->gmail;

        }
        ptr=ptr->next;
    }
}
void dlist::searchbygmail(char gm[20])
{
    ptr=head;
    while(ptr!=NULL)
    {
        if(strcmp(gm,ptr->gmail)==0)
        {
            cout<<"G-MAIL FOUND\n"<<endl;
            cout<<"CONTACT DETAILS ARE BELOW:\n"<<endl;
        cout<<"\n\tNAME ::\t"<<ptr->name;
        cout<<"\nNUMBER::\t+91-"<<ptr->number;
        cout<<"\nG-MAIL::\t"<<ptr->gmail;

        }
```

```
            ptr=ptr->next;
        }
    }
    void dlist::update(char n[20])
    {
        char ans;
        int c;
        ptr=head;
        while(ptr!=NULL)
        {
            if(strcmp(n,ptr->name)==0)
            {

                do
                {
                    cout<<"\nWHAT DO YOU WANT TO UPDATE?\n1.NAME\n2.PHONE
NUMBER\n3.G-MAIL\n";
                cin>>c;
                switch(c)
                {
                    case 1:
                    cout<<"ENTER NEW-NAME=";
                    cin>>ptr->name;
                    break;
                    case 2:
                    cout<<"ENTER NEW PHONE-NUMBER?";
                    cin>>ptr->number;
                    while(strlen(ptr->number)!=10)
    {
    cout<<"ENTER VALID NUMBER :";
    cin>>ptr->number;
    }
                    break;
                    case 3:
                    cout<<"ENTER NEW G-MAIL";
                    cin>>ptr->gmail;
                    break;
                }
                cout<<"DO YOU WANT TO CONTINUE UPDATING?";
```

```
                cin>>ans;
                }while(ans=='y');
            }
            ptr=ptr->next;
        }
    }
int main()
{
    char n[20];
    char nam[20];
    char name[10];
    char number[10];
    char gmail[20];
    dlist d1;
    // apply d;
    char ans;
    int ch,a;
    cout<<"************** PHONE BOOK ********************";
    cout<<"\n\nWHAT IS YOUR NAME?\n";
    cin.getline(name,20);
    cout<<"\n\n!!!!!!!!!!!!!!!!!!!! WELCOME "<<name<<"
!!!!!!!!!!!!!!!!!!!!";
    cout<<"\n\n\nLET'S CREATE OUR PHONEBOOK "<<name<<" \n\n";
    d1.accept();
    d1.sort();
    do
    {
    cout<<"\n\n\n\n1) DISPLAY YOUR PHONE BOOK\n2) INSERT NEW CONTACT\n3)
UPDATE DETAILS ON EXISTING CONTACT\n4) DELETE CONTACT\n5) SEARCH\n";
cin>>ch;
    switch(ch)
    {
    case 2:
    d1.insert();
    d1.sort();
    break;

    case 1:
    // d1.sort();
```

```
    d1.display();
    break;
    case 3:

        cout<<"\n\t ENTER THE NAME OF PERSON WHOSE DETAILS YOU WANT TO
UPDATE...\n";
        cin>>n;
    d1.update(n);
    d1.sort();
    break;
    case 4:
    cout<<"\t ENTER THE NAME YOU WANT TO DELETE FROM PHONEBOOK\t";
    cin>>name;
    d1.deletecontact(name);
    break;
    // case 5:
    // d1.deletesamename();
    // d1.display();
    // break;
    // case 6:
    // d1.deletesamenumber();
    // d1.display();
    // break;
    case 5:
    do
    {
    cout<<"1.SEARCH BY NAME\n2.SEARCH BY NUMBER\n3.SEARCH BY GMAIL";
    cin>>a;
    switch(a)
    {
        case 1:
        cout<<"ENTER THE NAME TO BE SEARCHED\n";
        cin>>name;
        d1.searchbyname(name);
        break;
        case 2:
        cout<<"ENTER THE NAME TO BE SEARCHED\n";
        cin>>number;
        d1.searchbynumber(number);
```

```
        break;
         case 3:
        cout<<"ENTER THE NAME TO BE SEARCHED\n";
        cin>>gmail;
        d1.searchbygmail(gmail);
        break;
        default:cout<<"\nNO PROPER INPUT GIVEN.....\n";
    }
    cout<<"DO YOU WANT TO CONTINUE SEARCHING?????????";
    cin>>ans;
}while(ans=='y');

    break;
    case 8:d1.deletesamegmail();
    d1.display();
    break;
    default:cout<<"\nNO PROPER INPUT GIVEN..\n";
    }
    cout<<"\n\nDO YOU WANT TO CONTINUE OPERATIONS?????????";
    cin>>ans;
}while(ans=='y');
}
```