TELEPHONE DIRECTORY

Prepared by :

Urlana Suresh Kumar

github.com/usk2003

Index

1.Abstract

Data structures in C are used to store data in an organized and efficient manner. The C programming language has many data structures, and the programmer can select any appropriate data structure and use it according to their convenience.

Some of the frequently used data structures are:

- arrays
- linked lists
- stacks
- queues
- trees
- graphs

A **Linked List** is a data structure. It is linear. The Linked List is like an array, but the Linked List is not stored sequentially in the memory. Every linked list has 2 parts, the data section and the address section that holds the address of the next element in the list, which is called a node.
The size of the linked list is not fixed, and data items can be added at any location in the list. The disadvantage is that to get to a node, we must traverse from the first node to the node that we require.
There are three types of link lists:

- Singly link list
- Circular linked list
- Double linked list

A **priority queue** is a special type of queue in which each element is associated with a priority value. And elements are served on the basis of their priority. That is, higher priority elements are served first.
However, if elements with the same priority occur, they are served according to their order in the queue.

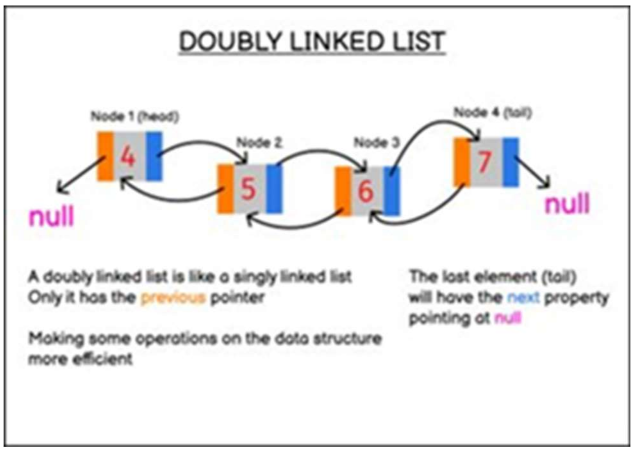
2.Introduction

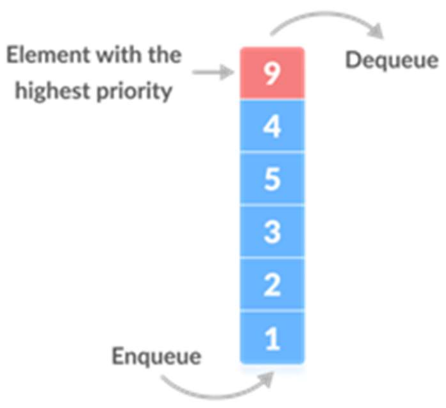
<u>**What is a telephone directory?**</u>

A **telephone directory**, commonly called a **telephone book**, **telephone address book**, **phone directory**, is a listing of telephone subscribers in a geographical area or subscribers to services provided by the organization that publishes the directory. Its purpose is to allow the telephone number of a subscriber identified by name and address to be found.
The advent of the Internet and smartphones in the 21st century greatly reduced the need for a paper phone book. Some communities, such as Seattle and San Francisco, sought to ban their unsolicited distribution as wasteful, unwanted and harmful to the environment.

<u>**What data structures are we using?**</u>
The Data Structure we are going to implement in this project is **Doubly Linked Lists.**
Important operations which are to be performed on the Phone Directory to make it organized are
•Sorting
•Searching
•Deleting
These operations can be easily performed using Doubly Linked Lists as all these operations can be done in either **Linear or Logarithmic** time complexities as follows:
•Sorting : O(n*Log n)
•Searching : O(n)
•Deletion : O(n)
we'll also be using priority queues, whose time complexities is as follows :

- insertion : O(n)
- Deletion : O(1)
- Peek : O(1)


DOUBLY LINKED LIST
Node 1 (head)
Node 2
Node 3
Node 4 (tail)
4
5
6
7
null
null
A doubly linked list is like a singly linked list
Only it has the previous pointer
Making some operations on the data structure more efficient
The last element (tail) will have the next property pointing at null



Why we chose doubly linked lists and priority queues :
Linked Lists are convenient because it is easy and fast to insert and remove frequently modified items from the list.
Linked list is a dynamic data structure and hence efficient memory utilization, i.e no need to pre-allocate memory.
A doubly-linked list is particularly preferred because you can iterate and search from the front (head) of the end of the list or the back (tail).
We used priority queues to prioritize certain contacts over the others and group selected contacts as favorites.

3.Objectives

The **Operations which we're implementing** in Phone Directory are :

1. Display the contact list
2. Add a new contact
3. Delete contacts with same name
4. Delete contacts with same number
5. Delete contacts with same gmail
6. Searching a contact by name, number or gmail
7. update a contact
8. Displaying the details of a particular contact
9. create a favorite contact list
10. display the favorite contact list

Details of a particular contact are stored in the data part of the Doubly Linked List. According to the user's choice, a corresponding algorithm can be applied to meet the user's needs for the final output.

4.Program

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct node
{
char number[100];
char gmail[100];
char name[100];
struct node *prev, *next;
} *head = NULL;
struct pq
{
char number[100];
char gmail[100];
char name[100];
int priority;
struct pq *previous, *nextt;
} *top = NULL;
void createFav()
{
char name[100];
int priority;
char ans;
do
{
printf("Enter the Name of the Contact and Assign a Priority
Number\n");
scanf("%[^\n]", name);
fflush(stdin);
scanf("%d", &priority);
fflush(stdin);
struct node *tmp = head;
struct pq *p, *temp1, *pr;
while (tmp != NULL)
{
if (strcmp(name, tmp->name) == 0)
{
p = (struct pq *)malloc(sizeof(struct pq));
strcpy(p->gmail, tmp->gmail);
strcpy(p->number, tmp->number);
strcpy(p->name, tmp->name);
p->priority = priority;
if (top == NULL)
{
p->nextt = p->previous = NULL;
p->priority = 1;
top = p;
}
else
{
temp1 = top;
while ((temp1->priority < priority) && (temp1->nextt != NULL))
{
```

```
temp1 = temp1->nextt;
}
if (temp1->nextt != NULL)
{
temp1->previous->nextt = p;
p->previous = temp1->previous;
p->nextt = temp1;
temp1->previous = p;
}
else
{
temp1->nextt = p;
p->nextt = NULL;
p->previous = temp1;
}
}
break;
}
tmp = tmp->next;
}
printf("Do You Want To Continue Assigning Priorities?\n");
scanf("%c", &ans);
fflush(stdin);
} while (ans == 'y');
}

void displayFav()
{
struct pq *temp = top;
printf("Displaying Favourite Contacts:\n");
while (temp != NULL)
{
printf("\nName ::\t%s\n", temp->name);
printf("Contact Number ::\t%s\n", temp->number);
printf("G-Mail ID :: \t%s\n", temp->gmail);
temp = temp->nextt;
}
}
void insert()
{
char number[100];
char gmail[100];
char name[100];
char ans;
do
{
struct node *p;
p = (struct node *)malloc(sizeof(struct node));
printf("Enter The Name of The Contact: ");
scanf("%[^\n]", name);
fflush(stdin);
strcpy(p->name, name);
printf("Enter The Phone Number    :\t");
scanf("%[^\n]", number);
fflush(stdin);
```

```
while (strlen(number) != 10)
{
printf("ENTER A VALID 10 DIGIT NUMBER  :\t");
scanf("%[^\n]", number);
fflush(stdin);
}
strcpy(p->number, number);
printf("Enter The G-Mail ID  :\t");
scanf("%[^\n]", gmail);
strcpy(p->gmail, gmail);
fflush(stdin);

if (head == NULL)
{
p->next = NULL;
p->prev = NULL;
head = p;
}
else
{
struct node *tmp = head;
while (tmp->next != NULL)
{
tmp = tmp->next;
}
p->next = NULL;
p->prev = tmp;
tmp->next = p;
}
printf("Do You Want To Continue Insertion?\n");
scanf("%c", &ans);
fflush(stdin);
} while (ans == 'y');
}
void display()
{
struct node *tmp = head;
while (tmp != NULL)
{
printf("\nNAME  ::  %s", tmp->name);
printf("\nNUMBER::  +91-%s", tmp->number);
printf("\nG-MAIL::  %s\n", tmp->gmail);
tmp = tmp->next;
}
}
void sort()
{
struct node *i, *j;
int temp;
char a[100], b[100], c[100];
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(a, i->name);
strcpy(i->name, j->name);
strcpy(j->name, a);
strcpy(b, i->number);
strcpy(i->number, j->number);
strcpy(j->number, b);
strcpy(c, i->gmail);
strcpy(i->gmail, j->gmail);
strcpy(j->gmail, c);
}
}
}
}
void deletecontact()
{
char s[100];
printf("Enter the Name of the Contact You Wish to Delete\n");
scanf("%[^\n]", s);
fflush(stdin);
int c = 0;
struct node *tmp = head;
while (tmp != NULL)
{
if (strcmp(s, tmp->name) == 0)
{
c = 1;
break;
}
else
{
c = 2;
}
tmp = tmp->next;
}
if (!tmp)
{
printf("CONTACT NOT FOUND\n");
return;
}
if (c == 1 && tmp != head && tmp->next != NULL)
{
tmp->prev->next = tmp->next;
tmp->next->prev = tmp->prev;
free(tmp);
printf("YOUR CONTACT IS SUCCESSFULLY DELETED\n\n");
}
if (tmp == head)
{
head = head->next;
head->prev = NULL;
free(tmp);
printf("YOUR CONTACT IS SUCCESSFULLY DELETED\n\n");
}
if (tmp->next == NULL)
{
```

```
tmp->prev->next = NULL;
tmp->prev = NULL;
free(tmp);
printf("YOUR CONTACT IS SUCCESSFULLY DELETED\n\n");
}
}
void deletesamename()
{
struct node *tmp1 = head;
int n = 0;
while (tmp1 != NULL && tmp1->next != NULL)
{
struct node *tmp2 = tmp1;
while (tmp2->next != NULL)
{
if (strcmp(tmp1->name, tmp2->next->name) == 0)
{
struct node *dup = tmp2->next;
tmp2->next = tmp2->next->next;
free(dup);
n++;
}
else
{
tmp2 = tmp2->next;
}
}
tmp1 = tmp1->next;
}
if (!n)
printf("Same Names Not Found\n");
else
display();
}
void deletesamegmail()
{
struct node *tmp1 = head;
int n = 0;
while (tmp1 != NULL && tmp1->next != NULL)
{
struct node *tmp2 = tmp1;
while (tmp2->next != NULL)
{
if (strcmp(tmp1->gmail, tmp2->next->gmail) == 0)
{
struct node *dup = tmp2->next;
tmp2->next = tmp2->next->next;
free(dup);
n++;
}
else
{
tmp2 = tmp2->next;
}
}
tmp1 = tmp1->next;
```

```
}
if (!n)
printf("Same G-Mail IDs Not Found\n");
else
display();
}
void deletesamenumber()
{
struct node *tmp1 = head;
int n = 0;
while (tmp1 != NULL && tmp1->next != NULL)
{
struct node *tmp2 = tmp1;
while (tmp2->next != NULL)
{
if (strcmp(tmp1->number, tmp2->number) == 0)
{
struct node *dup = tmp2->next;
tmp2->next = tmp2->next->next;
free(dup);
n++;
}
else
{
tmp2 = tmp2->next;
}
}
tmp1 = tmp1->next;
}
if (!n)
printf("Same Numbers Not Found\n");
else
display();
}
void searchbyname(char *na)
{
struct node *tmp = head;
int n = 0;
while (tmp != NULL)
{
if (strcmp(na, tmp->name) == 0)
{
printf("NAME FOUND\n");
printf("CONTACT DETAILS ARE BELOW:\n");
printf("\n\nNAME  ::\t%s", tmp->name);
printf("\nNUMBER::\t+91-%s", tmp->number);
printf("\nG-MAIL::\t%s", tmp->gmail);
n++;
}
tmp = tmp->next;
}
if (!n)
printf("CONTACT NOT FOUND\n");
}
void searchbynumber(char *num)
{
```

```
struct node *tmp = head;
int n = 0;
while (tmp != NULL)
{
if (strcmp(num, tmp->number) == 0)
{
printf("NUMBER FOUND\n");
printf("CONTACT DETAILS ARE BELOW:\n");
printf("\n\nNAME  ::\t%s", tmp->name);
printf("\nNUMBER::\t+91-%s", tmp->number);
printf("\nG-MAIL::\t%s", tmp->gmail);
n++;
}
tmp = tmp->next;
}
if (!n)
printf("CONTACT NOT FOUND\n");
}
void searchbygmail(char *gm)
{
struct node *tmp = head;
int n = 0;
while (tmp != NULL)
{
if (strcmp(gm, tmp->gmail) == 0)
{
printf("G-MAIL FOUND\n");
printf("CONTACT DETAILS ARE BELOW:\n");
printf("\n\nNAME  ::\t%s", tmp->name);
printf("\nNUMBER::\t+91-%s", tmp->number);
printf("\nG-MAIL::\t%s", tmp->gmail);
n++;
}
tmp = tmp->next;
}
if (!n)
printf("CONTACT NOT FOUND\n");
}
void update()
{
char n[100];
char ans;
int c;
struct node *tmp = head;
printf("\nEnter the Name of the Contact that You Wish to
Update\n");
scanf("%[^\n]", n);
fflush(stdin);
while (tmp != NULL)
{
if (strcmp(n, tmp->name) == 0)
{

do
{
```

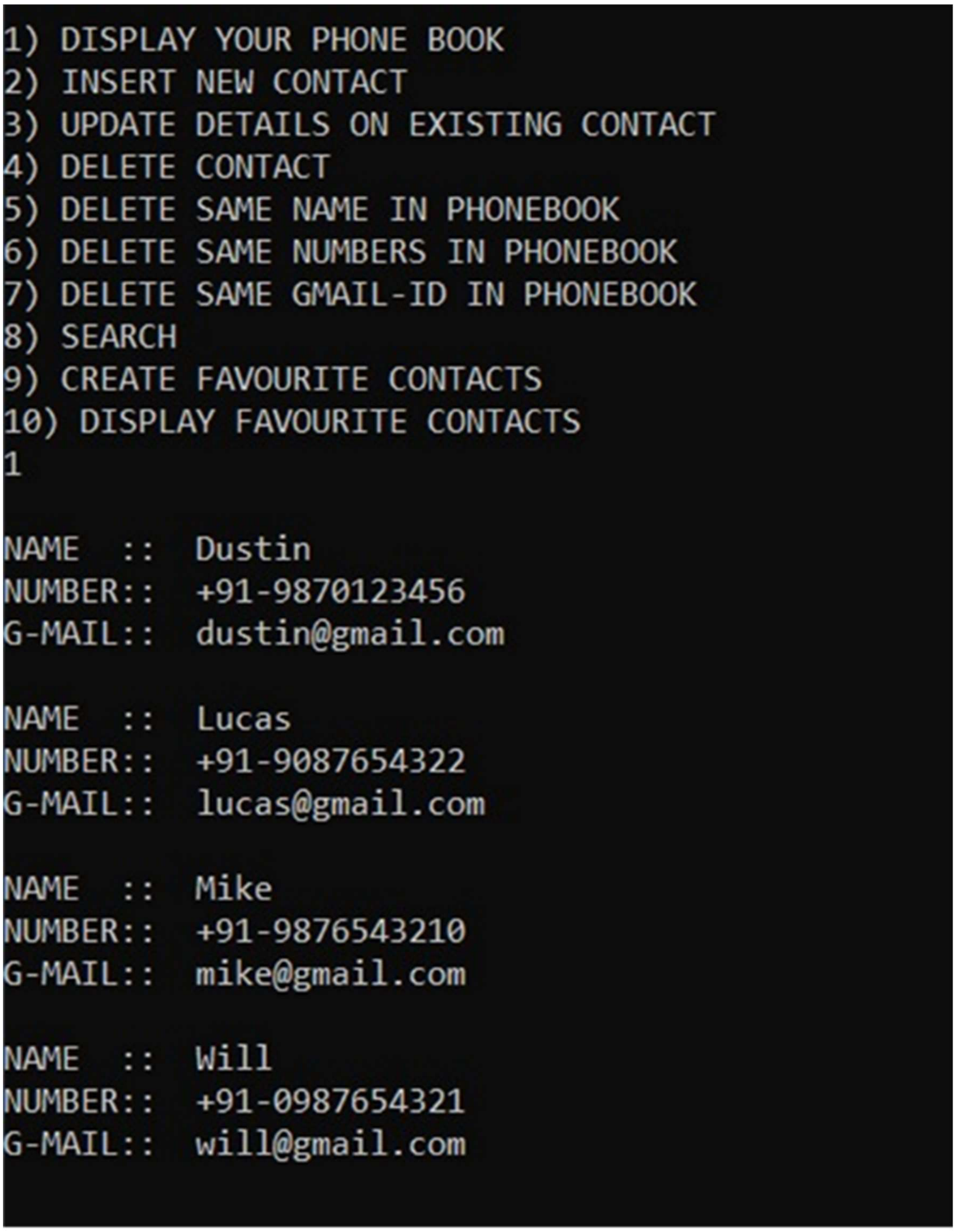
```
printf("\nWHAT DO YOU WANT TO UPDATE?\n1.NAME\n2.PHONE NUMBER\n3.G-
MAIL\n");
scanf("%d", &c);
fflush(stdin);
switch (c)
{
case 1:
printf("Enter The New Name:\n");
scanf("%[^\n]", tmp->name);
fflush(stdin);
break;
case 2:
printf("Enter the New Phone Number:\n");
scanf("%[^\n]", tmp->number);
fflush(stdin);
while (strlen(tmp->number) != 10)
{
printf("ENTER A VALID NUMBER!  :\n");
scanf("%[^\n]", tmp->number);
fflush(stdin);
}
break;
case 3:
printf("Enter The New G-Mail ID:\n");
scanf("%[^\n]", tmp->gmail);
fflush(stdin);
break;
}
printf("Do You Wish To Continue Updating?\n");
scanf("%c", &ans);
fflush(stdin);
} while (ans == 'y');
}
tmp = tmp->next;
}
}
int main()
{
char n[100];
char nam[100];
char name[100];
char number[100];
char gmail[100];
char ans;
int ch, a;
printf("*************************************  PHONE
BOOK  ************************************* \n");
printf("\n\nWHAT IS YOUR NAME?\n");
scanf("%[^\n]", name);
fflush(stdin);
printf("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  Welc
ome %s  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n",
name);
printf("\n!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! LET'S
CREATE OUR PHONEBOOK
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
```

```
insert();
sort();
fflush(stdin);
while (1)
{
printf("\n\n\n\n1) DISPLAY YOUR PHONE BOOK\n2) INSERT NEW
CONTACT\n3) UPDATE DETAILS ON EXISTING CONTACT\n4) DELETE
CONTACT\n5) DELETE SAME NAME IN PHONEBOOK\n6) DELETE SAME NUMBERS IN
PHONEBOOK\n7) DELETE SAME GMAIL-ID IN PHONEBOOK\n8) SEARCH\n9)
CREATE FAVOURITE CONTACTS\n10) DISPLAY FAVOURITE CONTACTS\n");
scanf("%d", &ch);
fflush(stdin);
switch (ch)
{
case 1:
display(); // Displays the contact list
break;

case 2:
insert(); // Inserts a new contact to the existing contact list
sort();
break;
case 3:
update(); // Updates the information needed of the given contact
sort();
break;
case 4:
deletecontact(); // Deletes contact with the given name
break;

case 5:
deletesamename(); // Deletes contacts with duplicate name other than
the contact with first entry with the given name
break;

case 6:
deletesamenumber(); // Deletes contacts with duplicate number other
than the contact with first entry with the given number
break;
case 7:
deletesamegmail(); // Deletes contacts with duplicate G-Mail ID
other than the contact with first entry with the given G-Mail ID
break;
case 8:
do
{
printf("1.SEARCH BY NAME\n2.SEARCH BY NUMBER\n3.SEARCH BY
GMAIL\n");
scanf("%d", &a);
fflush(stdin);
switch (a)
{
case 1:
printf("\nENTER THE NAME TO BE SEARCHED\n");
scanf("%[^\n]", name);
fflush(stdin);
```

```
searchbyname(name);  // Searches contact based on the name
break;
case 2:
printf("ENTER THE NUMBER TO BE SEARCHED\n");
scanf("%[^\n]", number);
fflush(stdin);
searchbynumber(number); // Searches contact based on the number
break;
case 3:
printf("ENTER THE GMAIL ID TO BE SEARCHED\n");
scanf("%[^\n]", gmail);
fflush(stdin);
searchbygmail(gmail); // Searches contact based on the G-Mail ID
break;
default:
printf("\nNO PROPER INPUT GIVEN.....\n");
}
printf("\nDO YOU WANT TO CONTINUE SEARCHING?????????\n");
scanf("%c", &ans);
fflush(stdin);
} while (ans == 'y');

break;

case 9:
createFav();
break;
case 10:
displayFav();
break;
default:
printf("\nENTER A VALID INPUT\n");
break;
}

printf("\n\nDO YOU WANT TO CONTINUE OPERATIONS?????????\n");
scanf("%c", &ans);
if (ans != 'y')
break;
fflush(stdin);
}
}
```

5. Results and Discussions

OUTPUT : CASE 1



```
1) DISPLAY YOUR PHONE BOOK
2) INSERT NEW CONTACT
3) UPDATE DETAILS ON EXISTING CONTACT
4) DELETE CONTACT
5) DELETE SAME NAME IN PHONEBOOK
6) DELETE SAME NUMBERS IN PHONEBOOK
7) DELETE SAME GMAIL-ID IN PHONEBOOK
8) SEARCH
9) CREATE FAVOURITE CONTACTS
10) DISPLAY FAVOURITE CONTACTS
1

NAME  ::  Dustin
NUMBER::  +91-9870123456
G-MAIL::  dustin@gmail.com

NAME  ::  Lucas
NUMBER::  +91-9087654322
G-MAIL::  lucas@gmail.com

NAME  ::  Mike
NUMBER::  +91-9876543210
G-MAIL::  mike@gmail.com

NAME  ::  Will
NUMBER::  +91-0987654321
G-MAIL::  will@gmail.com
```

6.Applications

A telephone directory is one of the most useful tools of information in communication. They provide contact info. You can use a directory to get emergency help.

Telephone directories have been an integral part of most public and academic libraries for nearly a century. Telephone directories represent an anomaly among library collections; known to virtually all users, they nevertheless often go unrecognized when librarians discuss reference sources.

Directories provide useful geographical information. Apart from contact information, a telephone directory is a useful resource when it comes to geographical details of a certain area. A local directory provides the transit map of the particular area or region.

You can use a directory to market your business. The contact information given such as the location, telephone numbers and postal addresses also help clients get easy access.

For advanced advertising, you can pay up for printed business adverts to appear in a telephone directory. This is a good tool of marketing since unlike newspapers and magazines which are discarded after reading; a directory is retained and is always within users' reach. Having your business listed in a directory also increases its credibility especially when it comes to investors and creditors.