# ABSTRACT

## BANK MANAGMENT SYSTEM:

It provides a simple interface for maintenance of bank information. It can be used by Banks to maintain the records of users easily. The Bank Management system is a management system which contains adding account, deleting account, depositing money and withdrawing money. Bank Management System would be able to maintain information and able to keep records of that particular event. This project can be implemented in any bank by fulfilling basic requirements. This Bank Management System will provide the transaction going inside the bank without manual processing. All information will be updated automatically by using the information stored in the system files.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTERS**  **CHAPTER** | **TITLE** | **PAGE NO** |
| **NO** |  |  |
|  | Acknowledgement | 3 |
|  | Abstract | 4 |
| 1 | Introduction | 6 |
| 2 | Analysis | 7 |
| 2.1 | Objectives of the Project | 7 |
| 2.2 | Requirement Specification | 7 |
| 3 | Implementation | 8 |
| 3.1 | Data Structures | 8 |
| 4 | Application | 15 |

|  |  |  |
| --- | --- | --- |
| 5 | Source code | 16 |
| 5.1 | Sample Output | 26 |
| 5.2 | Conclusion | 30 |
|  | Reference | 30 |

**Chapter 1**

## INTRODUCTION:

* The whole program is based on Bank Management and making the Bank work easier. Which also helps the user to easily create there Bank Account or delete their existing account.
* The target is to make the user work easier to access the bank problems. By this Bank Management System is one project which can deal with all the problems.
* The Bank Management system is an environment where all the process of the users in the Bank is managed. It is done through the automated computerized method. Conventionally this system is done using papers, files, and binders.
* As the number of the users increases in the Bank manually managing the strength becomes a hectic job for the administrator. This computerized system stores all the data in the database which makes it easy to fetch and update whenever needed.
* It includes processes like registration of the user’s details, depositing money, withdrawing money, and maintenance of the record. This system reduces the cost and workforce required for this job. As the system is online the information is globally present to everyone.

# CHAPTER 2

## ANALYSIS:

* 1. **OBJECTIVE OF THE PROJECT:**
* Adding or creating a account
* Deleting a Account
* Depositing Money
* Withdraw Money
* Transaction details

## Requirement Specification

Software Requirements:

* + - C COMPILER (Turbo C)

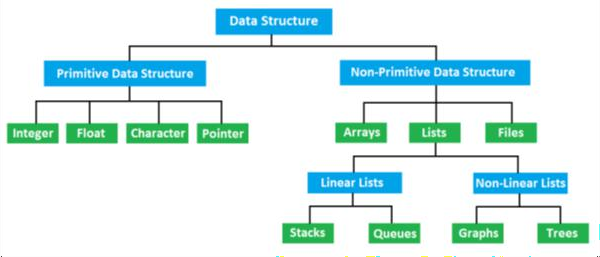
Hardware Requirements:

* + - Hard Disk – 2 GB
    - RAM Required – 2 GB

# CHAPTER 3

## IMPLEMANTATION

**3.1 DATA STRUCTURE**



Data is a collection of elements and structure is the organization of these elements. Data structure is the study of

How effectively data is collected and stored. How effectively data is organized.

How effectively data is retrieved.

How effectively logical relations are found.

Data structures are of different structures being primitive data structures and non-primitive data structures: Primitive data structures are data structures which can be directly manipulated using the machine instructions.

Some examples of **primitive data structures** are:

. Int

. Float

. Char

. Pointer

**Non-primitive data structures** are the data structures which cannot be manipulated directly using the machine instructions.

Some exampled of non-primitive data structures are:

. Arrays

. Lists

. Files

Lists are further classified into linear and non-linear lists. Linear lists consist of:

. Stack

. Queue

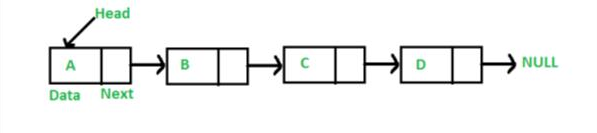
. Linked list

Non-linear lists consist of:

. Trees

. Graph

The data structure implemented in this program is linked list:



## Linked List

A linked list is a linear data structure which consists of different data elements called nodes.

Each node will have 2 parts. The first part consists of data or some value while the second part holds the address of the next node.

Memory is allocated to the nodes using dynamic memory allocation functions such **malloc(), calloc(), realloc() and free().**

1. **malloc() :** This function is used to allocate a complete single block of memory of the specified size. It returns the starting address of the block of memory allocated and hence a pointer is used to store the address returned by malloc.

Syntax –

datatype \*ptr = (datatype \*)malloc(size)

1. **calloc() :** This function allocates the specified size of memory in multiple block having the same size. Each block is assigned to NULL. It returns the starting address of the block of memory allocated and hence a pointer is used to store the address returned.

Syntax -

datatype \*ptr = (datatype \*)calloc(size , number of blocks)

1. **realloc() :** This function is used to reallocate the allocated memory. It returns the starting address of the new block of memory allocated and hence a pointer is used to store the address returned

Syntax -

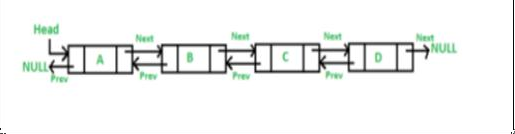
datatype \*ptr = (datatype \*)realloc(ptr , size)

ptr is the name of the pointer storing the address of the memory block.

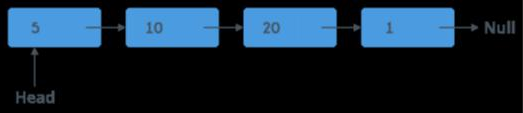
1. **free() :** This function is used to free the allocated memory. The parameter passed in free() is the pointer storing the address of the allocated memory block.

Syntax – free(pointer name)

There are 4 different types of linked lists:



## Single Linked List (SLL)



A single linked list is a linked list which consists of a group of nodes where each node has 2 parts. One part is the data stored in the linked list and the other is the address of the next node. The address part of the last node always stores the value NULL. A pointer (Head in above figure)

points to the first node of the linked list and is used to access the linked list.

## Double Linked List (DLL):

A double linked list is a linked list which consists of a group of nodes where each node has 3 parts. One part is the data stored in the linked list, the second part is the address of the next node and the third part is the address of the previous node. The next address part of the last node points to NULL and similarly the previous address part of the first node points to NULL. Doubly linked list, despite of occupying more memory than singly linked lists, are more commonly used as you can traverse from the first node to the last and vice versa, making it easier to access the nodes.

## Circular Linked List (CLL)

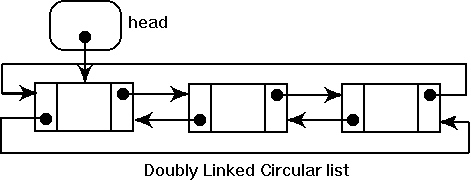
A circular linked is a list in which the address part of the last node stores the address of the first node. Circular linked lists can be single circular linked lists or double circular linked lists.

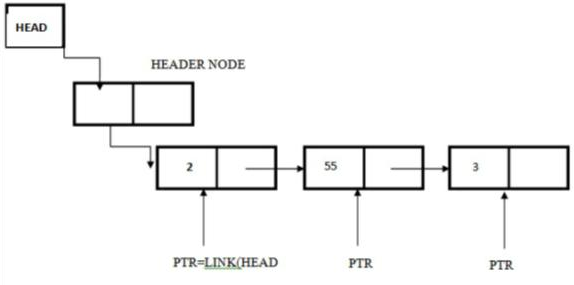
Singly Circular Linked Lists :

The last node points the first node.

## Circular Linked Lists :

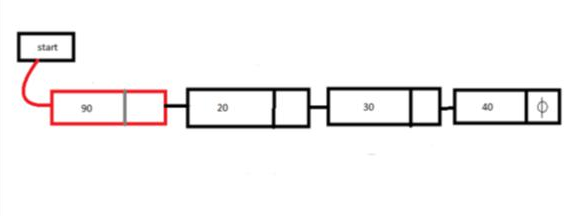
The last node points to the first node and the first node points to the last.





## 4. Header Linked List (HLL):

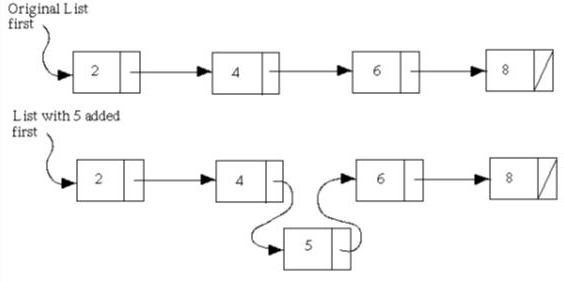
Header linked list is a linked list in which an extra node called the header node is present between the head node and the first node of the linked list. The header node does not need to store any data. It can be used as a reference node or be left blank. The header node can also contain the sum of all the elements stored in the linked list. Header node (in red) contains 90 which is sum of all other elements.



## Operations on Linked Lists :

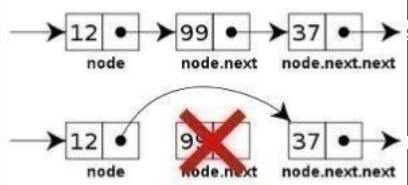
**INSERTION**

1. **Insertion at beginning:** A node can be added in the beginning of the List. In this, the head points to the new node and the new node is made to point to the node to which the address previously pointed.
2. **Insert at end**: In this, we traverse up to the last node and then the node at the end is made to point to the new node and the new node is pointed to NULL.
3. **Insertion at kth position:** In this we traverse up to the node at k-1 and make this node point to the new node and make the new node point to the node at which the (k-1) th node pointed.
4. **Insertion before/after key:** In this we traverse up to the node having the key value and according to the condition insert the new node before or after that node.



## DELETION

1. **Deletion at beginning:** In this we delete the node to which the head points and head is now made to point the the node at which the deleted node pointed.
2. **Deletion at end:** In this we traverse up to the second last node and make that node point to NULL and delete the last node i.e. the node to which this node points.
3. **Deletion at kth position**: In this we traverse up to the node at k-1 position and make it point to the node at k+1 position and delete the node to which the (k-1) th node previously pointed.
4. **Deletion before/after key:** In this we traverse up to the node having the key value and according to the condition delete the node before or after that node.



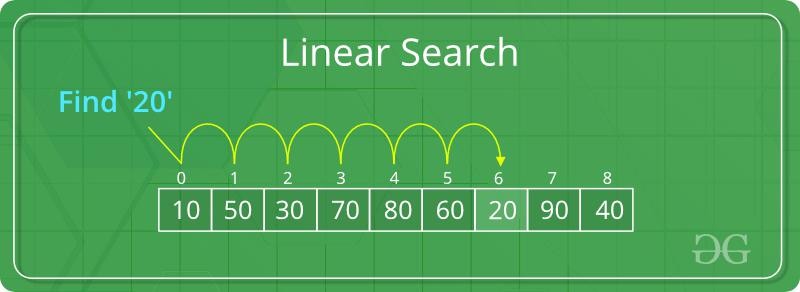
## SEARCH:

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

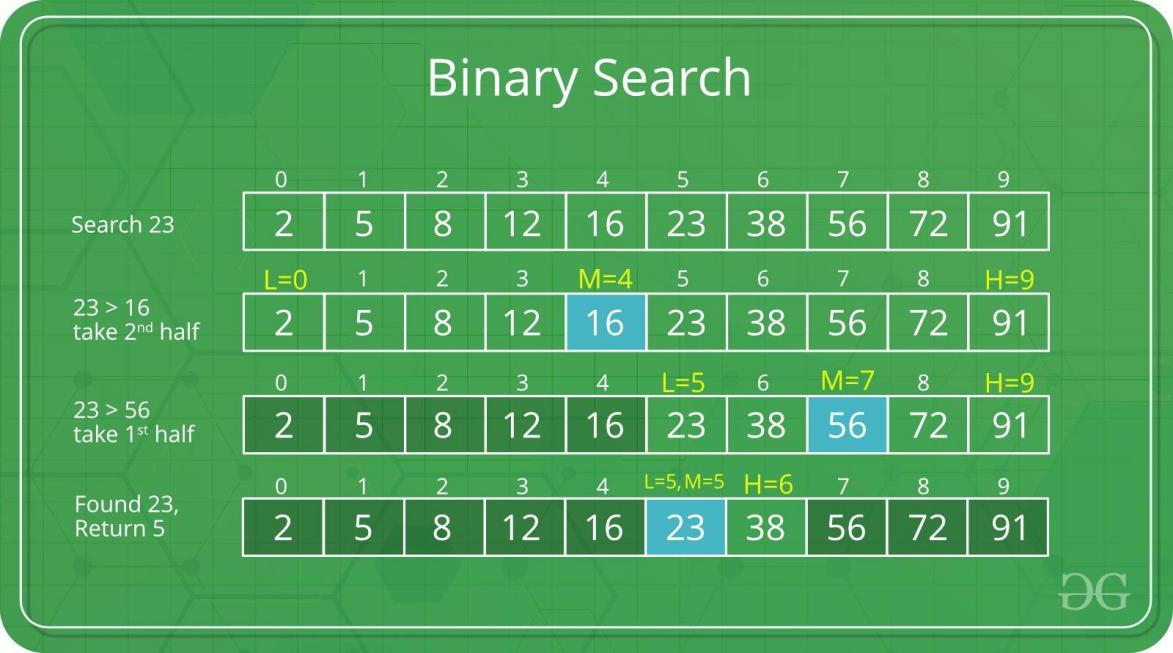
Sequential Search: In this, the list or array is traversed sequentially and every element is checked. For example: [Linear Search](https://www.geeksforgeeks.org/linear-search/).

Interval Search: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: [Binary Search](https://www.geeksforgeeks.org/binary-search/).

Linear Search to find the element “20” in a given list of numbers



Binary Search to find the element “23” in a given list of numbers



# CHAPTER 4

## APPLICATION:

* This project can be implemented in any bank by fulfilling basic requirements. This Bank Management System will provide the transaction going inside the bank without manual processing. All information will be updated automatically by using the information stored in the system files.
* Maintain records for long time.
* Fast and perfect.
* Time Management.
* Automatic upgradation of Information.

# CHAPTER 5 SOURCE CODE

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <time.h>

typedef struct sll

{

char money[50]; struct sll \*next\_node;

} sll;

void Account\_numgen(void); int check\_Account\_num(void);

void depositMoney(sll \*\*, int \*); void withdrawMoney(sll \*\*, int \*); void transaction(sll \*\*, char \*); void showBalance(int \*);

void display\_transaction(sll \*\*); void help();

int main()

{

int choice1, choice2;

int Account\_NumberValid , balance ;

sll \*head = NULL;

lable: Account\_NumberValid=0; balance=0;

while (1)

{

printf("\n\n----------ABHINAV BANK OF INDIA \n\n");

printf("1. Create Account \n2. Account info / Withdraw-add money\n3. Exit\n4. Help desk\n"); printf("\nYour choice: ");

scanf("%d", &choice1);

switch (choice1)

{

case 1:

Account\_numgen(); goto lable;

case 2:

Account\_NumberValid = check\_Account\_num();

if (Account\_NumberValid)

{

printf("\nValid Account\_Number\n");

}

else

{

printf("\nInvalid Account\_Number. Please create a Account by selecting option 1\n"); exit(0);

}

while (Account\_NumberValid)

{

printf("\n----------ABHINAV BANK OF INDIA MENU \n\n\n");

printf("1. Check Balance\n2. Deposit Money\n3. Withdraw Money\n4. View transaction details\n5.

Quit(Your Account will be deleted)\n6. Help desk\n7. Back\n");

printf("Enter choice: "); scanf("%d", &choice2);

switch (choice2)

{

case 1:

showBalance(&balance); break;

case 2:

depositMoney(&head, &balance); break;

case 3:

withdrawMoney(&head, &balance); break;

case 4:

display\_transaction(&head); break;

case 5:

printf("\nTHANL YOU FOR BANKING WITH US\n"); printf("HOPE TO SEE YOU AGAIN");

exit(0); case 6:

help(); break;

case 7:

goto lable; break;

default:

printf("\nInvalid option entered!\n"); break;

}

}

case 3:

printf("HOPE TO SEE YOU AGAIN THANK YOU FOR BANKING WITH US");

exit(0);

case 4:

help(); break;

default:

printf("\nInvalid choice...Try again \n");

break;

}

}

return 0;

}

void Account\_numgen(void)

{

FILE \*fp;

srand(time(NULL));

int generatedAccount\_Number = 1000 + rand() % 9000;

printf("\nAccount Created successfully\n");

printf("\nYour generated Account\_Number: %d\n", generatedAccount\_Number); printf("\nRe-run the program and use our bank with this Account\_Number\n\n");

fp = fopen("temp.txt", "a"); if (NULL == fp)

{

printf("\nCannot open file!"); exit(0);

}

fprintf(fp, "%d\n", generatedAccount\_Number); fclose(fp);

}

int check\_Account\_num(void)

{

FILE \*fp;

char Account\_Number[8]; char keyAccount\_Number[8]; int Account\_NumberValid = 0;

printf("\n\nEnter the Account\_Number: \n"); scanf("%s", keyAccount\_Number);

fp = fopen("temp.txt", "r");

if (NULL == fp)

{

printf("\nFile cannot be opened\n"); exit(0);

}

while (fgets(Account\_Number, sizeof(Account\_Number), fp) != NULL)

{

if (strstr(Account\_Number, keyAccount\_Number))

{

Account\_NumberValid = 1;

}

}

fclose(fp);

return Account\_NumberValid;

}

void showBalance(int \*balance)

{

printf("\nYour current balance is Rs.%d\n", \*balance);

}

void depositMoney(sll \*\*head, int \*balance)

{

int depositAmount;

char depositStmt[50];

printf("\nEnter amount to deposit: "); scanf("%d", &depositAmount);

if (depositAmount > 0)

{

\*balance += depositAmount;

printf("\nRs.%d deposited\n", depositAmount);

snprintf(depositStmt, sizeof(depositStmt), "Rs.%d deposited\n", depositAmount); transaction(head, depositStmt);

}

else

{

printf("\nInvalid amount entered\n.");

}

}

void withdrawMoney(sll \*\*head, int \*balance)

{

int withdraw\_amount; char withdrawStmt[50];

printf("\nEnter amount to withdraw: "); scanf("%d", &withdraw\_amount);

if (withdraw\_amount > 0)

{

if (withdraw\_amount > \*balance)

{

printf("\nCannot withdraw. Balance Rs.%d\n", \*balance);

}

else

{

\*balance = \*balance - withdraw\_amount; printf("\nRs.%d withdrawn\n", withdraw\_amount);

snprintf(withdrawStmt, sizeof(withdrawStmt), "Rs.%d withdrawn\n", withdraw\_amount); transaction(head, withdrawStmt);

}

}

else

{

printf("\nInvalid amount entered\n.");

}

}

void transaction(sll \*\*head, char \*str)

{

static int c = 0; sll \*temp;

temp = (sll \*)malloc(sizeof(sll));

strcpy(temp->money, str); temp->next\_node = NULL;

if (NULL == \*head)

{

\*head = temp;

c++;

}

else

{

sll \*p;

p = \*head;

while (NULL != p->next\_node)

{

p = p->next\_node;

}

p->next\_node = temp; c++;

}

}

void display\_transaction(sll \*\*head)

{

sll \*temp; temp = \*head;

if (NULL == temp)

{

printf("\nNo transaction history...\n");

}

else

{

printf("\nTransaction History\n \n\n");

while (NULL != temp)

{

printf("%s\n", temp->money); temp = temp->next\_node;

}

}

}

void help()

{

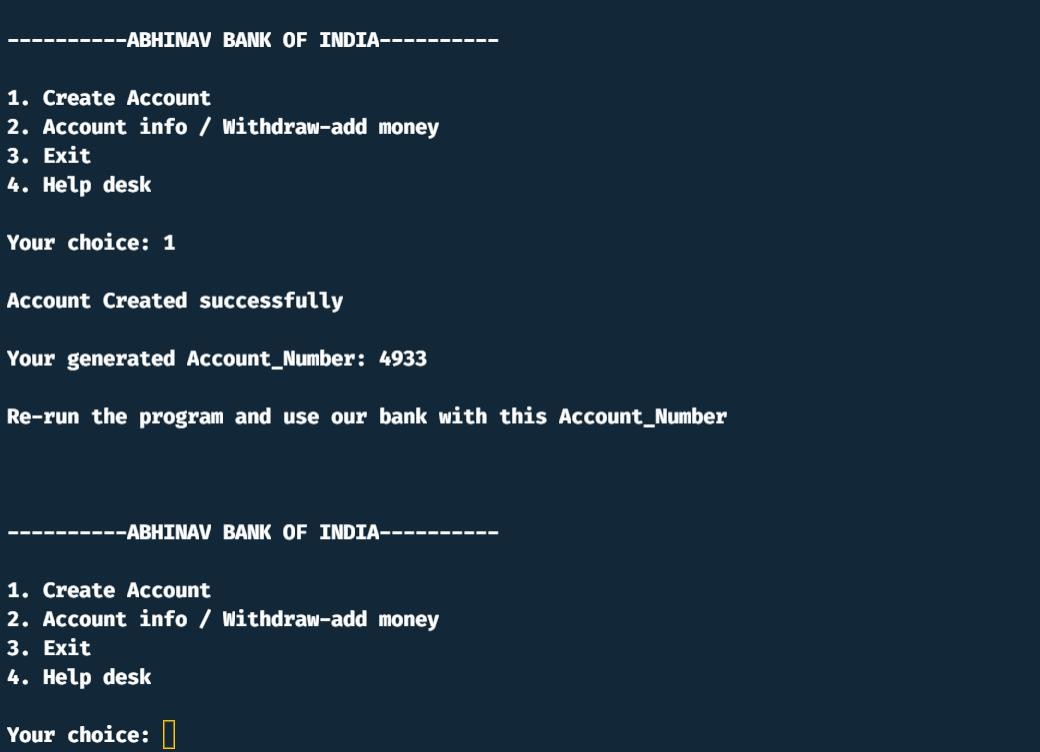
printf("\n\n\n\nYou can reach us by calling our toll free Number 7204881779 \n"); printf(" or ");

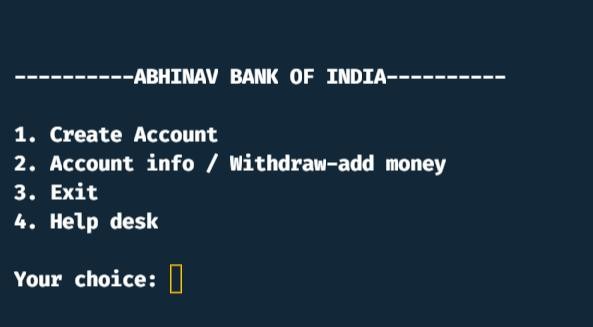
printf("\nYou can write us at abhinavdevraj@gmail.com\n"); printf(" or ");

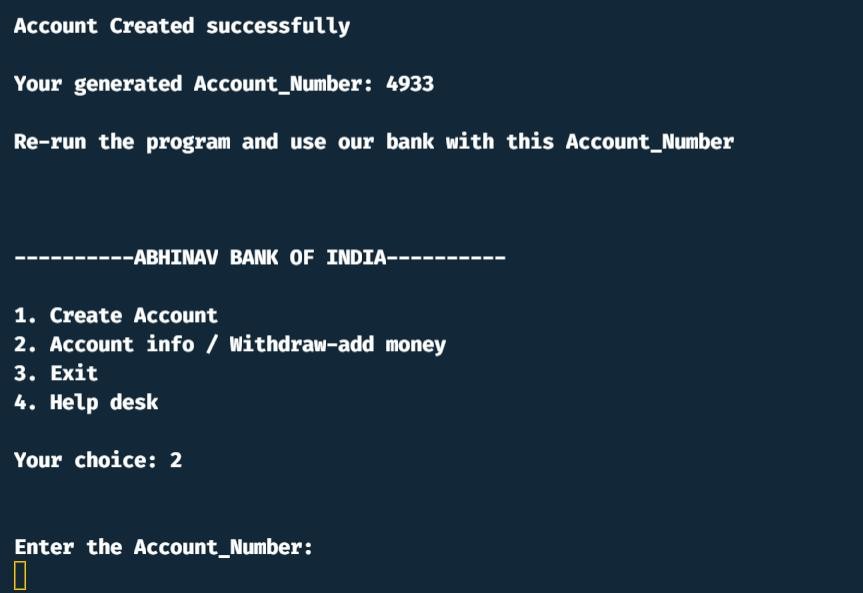
printf("\nYou can visit our nearest Bank\n\n\n\n\n\n\n\n"); exit(0);

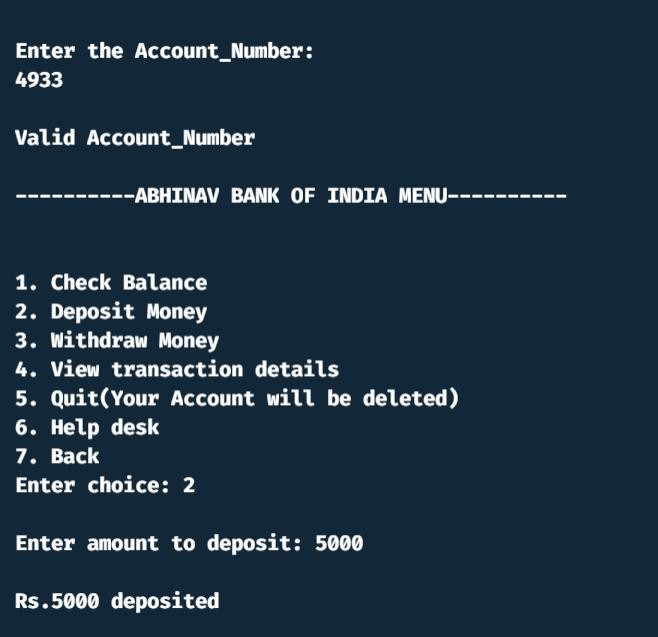
}

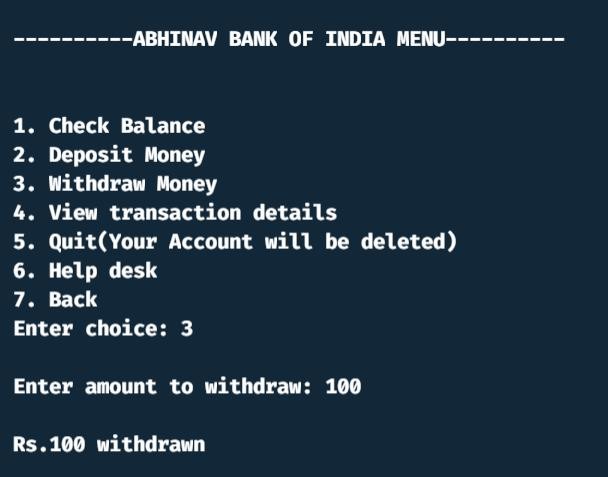
# SAMPLE OUTPUT:

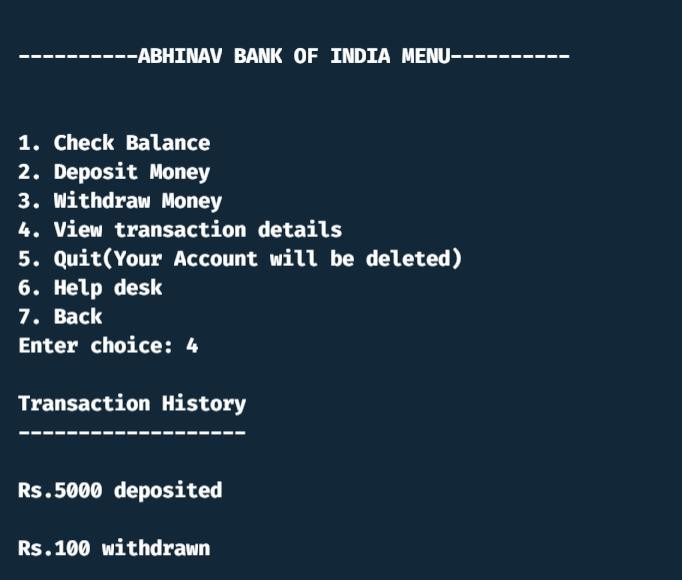


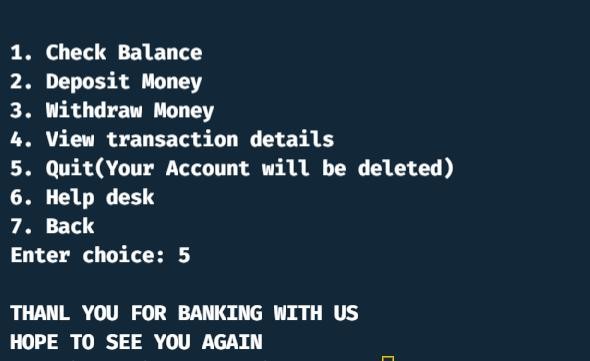












**5.2 CONCLUTION:**

Bank Management System would be able to maintain information and able to keep records of that particular event . This project can be implemented in any bank by fulfilling basic requirements. This Bank Management System will provide the transaction going inside the bank without manual processing. All information will be updated automatically by using the information stored in the system files.

# REFERENCES

[www.geeksforgeeks.org/data-structures/linked-list/](http://www.geeksforgeeks.org/data-structures/linked-list/)

[www.udemy.com/course/datastructurescncpp/learn/lecture/](http://www.udemy.com/course/datastructurescncpp/learn/lecture/)