Certificate

Abstract

The Loan Management System is a file structure system. In this file is created using loan related details. This uses variable length fields with fixed length records. Later, after adding records into the file, operation like reading the details, searching the record, deleting the record, and modifying a record operation can be performed on the contents of file.

This mini project is aimed at providing a common platform for all kinds of banks and loan agencies. Here, in order to search a record efficiently indexing is built on primary key value which is account number using hashing technique. Hashing uses add and fold algorithm and primary key values are hashed to particular location in a hash file and stored. In future records can be retrieved from the same location.

Acknowledgment

The fulfilment and rapture that go with the fruitful finishing of any assignment would be inadequate without the specifying the people who made it conceivable, whose steady direction and support delegated the endeavors with success.

I would like to profoundly thank **Management** of **RNS Institute of Technology** for providing such a healthy environment to carry out this Project work.

I would like to thank our beloved Director **Dr. H N Shivashankar** for his confidence filling words and support for providing facilities throughout the course.

I would like to express our thanks to our Principal **Dr. M K Venkatesha** for his support and for inspiring us towards the attainment of knowledge.

I wish to place on record our words of gratitude to **Dr. M V Sudhamani,** Professor and Head of the Department, Information Science and Engineering, for being the enzyme and master mind behind our Project work.

I would like to express our profound and cordial gratitude to our Faculty incharge **Mr. Santosh Kumar**, Assistant Professor, Department of Information Science and Engineering for her valuable guidance, constructive comments and continuous encouragement throughout the Project work.

I would like to express our profound and cordial gratitude to our guide **Dr. M V Sudhamani**, Professor and Head of the Department, Information Science and Engineering, for her valuable guidance in preparing Project report.

I would like to thank all other teaching and non-teaching staff of Information Science & Engineering who have directly or indirectly helped us to carry out the project work.

And lastly, I would hereby acknowledge and thank our parents who have been a source of inspiration and also instrumental in carrying out this Project work.

<div align="right">

CHIRAYU GUPTA

1RN16IS028

</div>

Abbreviation

ASCII   American Standard Code for Information Interchange

AVL   Adelson-Velskii and Landis

MDI   Menu Driven Interface

OS   Operating System

RAM   Random Access Memory

ACCNO   Account Number

UI   User Interface

Addr   Address

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1   Introduction to File Structures

File Structure is a combination of representations of data in files and operations for accessing the data. It allows applications to read, write and modify data. It supports finding the data that matches some search criteria or reading through the data in some particular order.

1.1.1   History

Early work with files presumed that files were on tape, since most files were. Access was sequential, and the cost of access grew in direct proportion, to the size of the file. As files grew intolerably large for unaided sequential access and as storage devices such as hard disks became available, indexes were added to files.

The indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With key and pointer, the user had direct access to the large, primary file. But simple indexes had some of the same sequential flaws as the data file, and as the indexes grew, they too became difficult to manage , especially for dynamic files in which the set of keys changes.

A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes. One use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. An example is finding similar stretches in DNA sequences. They are also useful in cryptography.

Hash functions are related to (and often confused with) checksums, check digits, fingerprints, lossy compression, randomization functions, error-correcting codes, and ciphers. Although these concepts overlap to some extent, each has its own uses and requirements and is designed and optimized differently. The Hash Keeper database maintained by the American National Drug Intelligence Center, for instance,

is more aptly described as a catalogue of file fingerprints than of hash values.

1.1.2 About the File

When we talk about a file on disk or tape, we refer to a particular collection of bytes stored there. A file, when the word is used in this sense, *physically* exists. A disk drive may contain hundreds, even thousands of these *physical files*.

From the standpoint of an application program, a file is somewhat like a telephone line connection to a telephone network. The program can receive bytes through this phone line or send bytes down it, but it knows nothing about where these bytes come from or where they go. The program knows only about its end of the line. Even though there may be thousands of physical files on a disk, a single program is usually limited to the use of only about 20 files.

The application program relies on the OS to take care of the details of the telephone switching system. It could be that bytes coming down the line into the program originate from a physical file they come from the keyboard or some other input device. Similarly, bytes the program sends down the line might end up in a file, or they could appear on the terminal screen or some other output device. Although the program doesn't know where the bytes are coming from or where they are going, it does know which line it is using. This line is usually referred to as the *logical file*, to distinguish it from the *physical files* on the disk or tape.

1.1.3 Various Kinds of storage of Fields and Records

A field is the smallest, logically meaningful, unit of information in a file.

Field Structures

The four most common methods as shown in Figure 1.1 of adding structure to files to maintain the identity of fields are:

- Begin each field with a length indicator.
- Place a delimiter at the end of each field to separate it from the next field.
- Use a "keyword=value" expression to identify each field and its contents.

Method 1: Fix the Length of Fields

In the above example, each field is a character array that can hold a string value of some maximum size. The size of the array is 1 larger than the longest string it can hold. Simple arithmetic is sufficient to recover data from the original fields.

The disadvantage of this approach is adding all the padding required to bring the fields up to a fixed length, makes the file much larger. We encounter problems when data is too long to fit into the allocated amount of space. We can solve this by fixing all the fields at lengths that are large enough to cover all cases, but this makes the problem of wasted space in files even worse. Hence, this approach isn't used with data with large amount of variability in length of fields, but where every field is fixed in length if there is very little variation in field lengths.

Method 2: Begin Each Field with a Length Indicator

We can count to the end of a field by storing the field length just ahead of the field. If the fields are not too long (less than 256 bytes), it is possible to store the length in a single byte at the start of each field. We refer to these fields as *length-based*.

Method 3:Separate the Fields with Delimiters

We can preserve the identity of fields by separating them with delimiters. All we need to do is choose some special character or sequence of characters that will not appear within a field and then *insert* that delimiter into the file after writing each field.   White-space characters (blank, new line, tab) or the vertical bar character, can be used as delimiters.

Method 4:Use a "Keyword=Value" Expression to Identify Fields

This has an advantage the others don't. It is the first structure in which a field provides information about itself. Such *self-describing structures* can be very useful tools for organizing files in many applications. It is easy to tell which fields are contained in a file.

```
Ames        John      123 Maple    Stillwater  OK74075377-1808
Mason       Alan      90 Eastgate  Ada         OK74820
```

(a) Field lengths fixed. Place blanks in the spaces where the phone number would go.

```
Ames|John|123 Maple|Stillwater|OK|74075|377-1808|
Mason|Alan|90 Eastgate|Ada|OK|74820||
```

(b) Delimiters are used to indicate the end of a field. Place the delimiter for the "empty" field immediately after the delimiter for the previous field.

```
Ames|…|Stillwater|OK|74075|377-1808|#Mason|… 90Eastgate|Ada|OK|74820|#…
```

(c) Place the field for business phone at the end of the record. If the end-of-record mark is encountered, assume that the field is missing.

```
SURNAME=Ames|FIRSTNAME=John|STREET=123 Maple|…|ZIP=74075|PHONE=377-1808|#…
```

(d) Use a keyword to identify each field. If the keyword is missing, the corresponding field is assumed to be missing.

Figure 1.1 Four methods for field structures

Even if we don't know ahead of time which fields the file is supposed to contain. It is also a good format for dealing with missing fields. If a field is missing, this format makes it obvious, because the keyword is simply not there. It is helpful to use this in combination with delimiters, to show division between each value and the keyword for the following field. But this also wastes a lot of space: 50% or more of the file's space could be taken up by the keywords.

A record can be defined as a set of fields that belong together when the file is viewed in terms of a higher level of organization.

Record Structures

The five most often used methods for organizing records of a file as shown in Figure 1.2 and Figure 1.3 are:

- Require the records to be predictable number of bytes in length.
- Require the records to be predictable number of fields in length.

- Begin each record with a length indicator consisting of a count of the number of bytes that the record contains.
- Use a second file to keep track of the beginning byte address for each record.
- Place a delimiter at the end of each record to separate it from the next record.

Method 1:Make the Records a Predictable Number of Bytes (Fixed-Length Record)

A *fixed-length record file* is one in which each record contains the same number of bytes. In the field and record structure shown, we have a fixed number of fields, each with a predetermined length, that combine to make a fixed-length record.

Fixing the number of bytes in a record does not imply that the size or number of fields in the record must be fixed. Fixed-length records are often used as containers to hold variable numbers of variable-length fields. It is also possible to mix fixed and variable-length fields within a record.

Method 2: Make Records a Predictable Number of Fields

Rather than specify that each record in a file contains some fixed number of bytes, we can specify that it will contain a fixed number of fields.
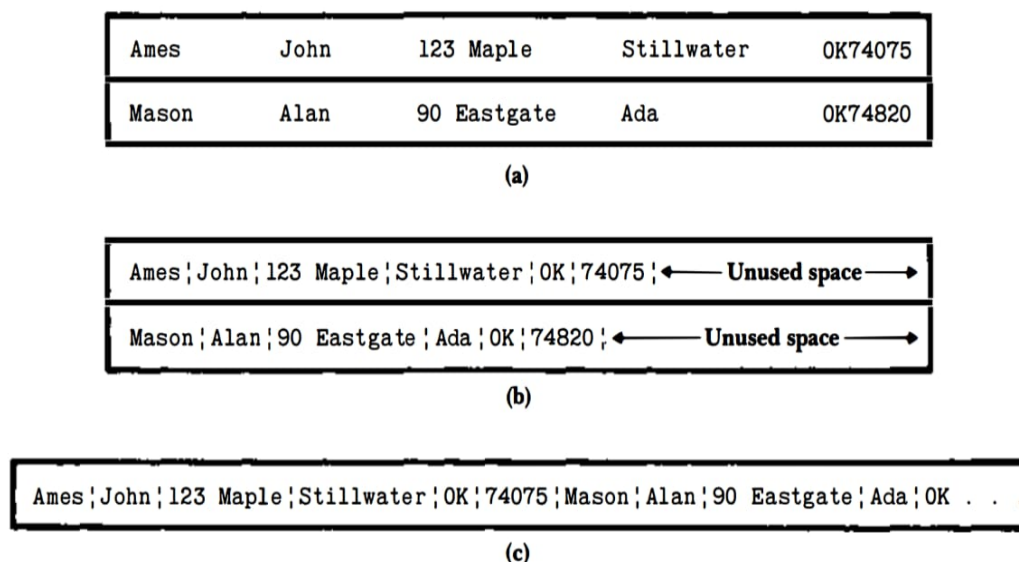
Figure 1.2 Making Records Predictable number of Bytes and Fields

Method 3: Begin Each Record with a Length Indicator

We can communicate the length of records by beginning each record with a filed containing an integer that indicates how many bytes there are in the rest of the record. This is commonly used to handle variable-length records.

Method 4: Use an Index to Keep Track of Addresses

We can use an index to keep a byte offset for each record in the original file. The byte offset allows us to find the beginning of each successive record and compute the length of each record. We look up the position of a record in the index, then seek to the record in the data file.

Method 5: Place a Delimiter at the End of Each Record

It is analogous to keeping the fields distinct. As with fields, the delimiter character must not get in the way of processing. A common choice of a record delimiter for files that contain readable text is the end-of-line character (carriage return/ new-line pair or, on Unix systems, just a new-line character: \n). Here, we use a # character as the record delimiter.
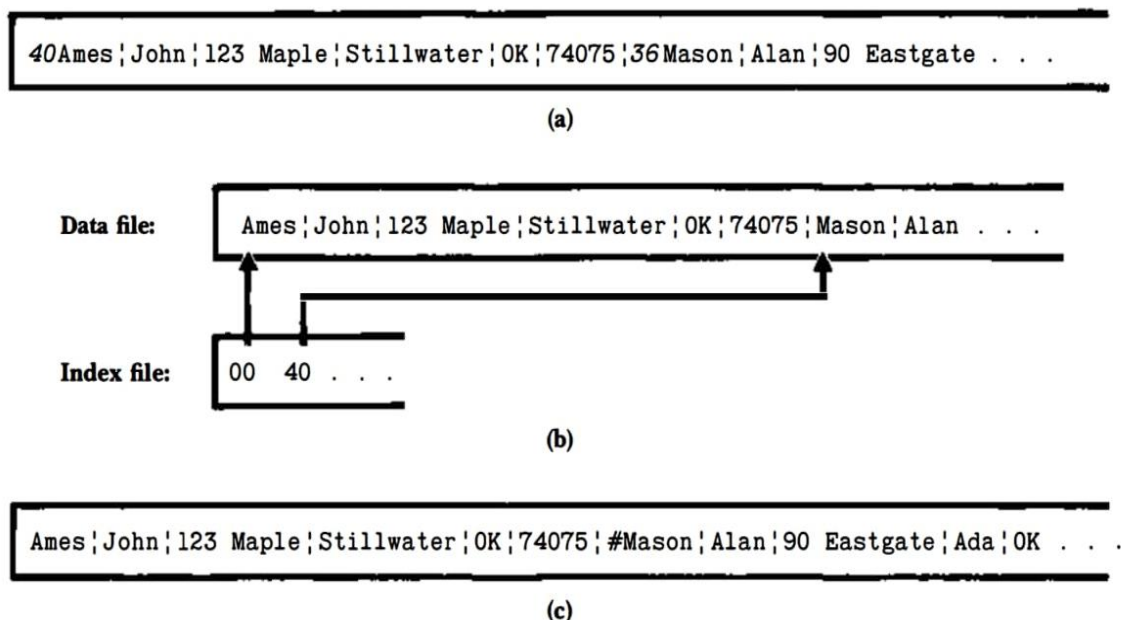
Figure 1.3 Using Length Indicator, Index and Record Delimiters

1.1.4   Application of File Structure

Relative to other parts of a computer, disks are slow. One can pack thousands of megabytes on a disk that fits into a notebook computer.

The time it takes to get information from even relatively slow electronic random access memory (RAM) about 120 nanoseconds. Getting the same information from a typical disk takes 30 milliseconds. So, the disk access is a quarter of a million times longer than a memory access. Hence, disks are *very* slow compared to memory. On the other hand, disks provide enormous capacity at much less cost than memory. They also keep the information stored on them when they are turned off.

Tension between a disk's relatively slow access time and its enormous, nonvolatile capacity, is the driving force behind file structure design. Good file structure design will give us access to all the capacity without making our applications   spend   a   lot   of   time   waiting   for   the   disk.

Chapter 2

SYSTEM ANALYSIS

2.1   Analysis of Application

In this application, we deal with loan agency in which the loan is given to the customer with respect to the type of loan the customer needs. There will be a particular rate of interest which will be provider for different type of loan. The customer need to enter certain details like their account number, name, E-mail ID, occupation, phone number and there will be three loan types which will be displayed from which the customer needs to choose one loan type. The customer has the opportunity of displaying all the records using hashing, adding records to the file ,searching, deleting, updating and they can even quit the application.

2.2   Structure used to Store the Fields and Records

Storing Fields

Fixing the Length of Fields:

In the Loan Management System, the ACCNO field is a character array that can hold a string value of some maximum size. The size of the array is larger than the longest string it can hold. Other fields are -

- Accno   Char [5]
- Name   Char [10]
- Addr   Char [10]
- email   Char [15]
- phno   Char [10]
- occupation   Char [10]
- loantype   Char [10]

Separating the Fields with Delimiters:

A delimiter is a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data streams. In this application,

delimiter used for records is \n and for fields is | .

Storing Records

Making Records a Predictable Number of Field:

In this system, we have a fixed number of fields, each with a maximum length, that combine to make a data record. Fixing the number of fields in a record does not imply that the size of fields in the record is fixed. The records are used as containers to hold a mix of fixed and variable-length fields within a record. We have 44 contiguous fields and we can recognize fields simply by counting the fields *modulo 44*.

Using an Index to Keep Track of Addresses:

We use a Hash key to keep byte offsets for each record in the original file. The byte offsets allow us to find the beginning of each successive record and compute the length of each record. We look up the position of a record in the Hash table, and then seek to the record in the data file.

Placing a Delimiter at the End of Each Record

Our choice of a record delimiter for the data files is the end-of-line (new-line) character

2.3   Operations Performed on a File

Insertion

The system is initially used to add customer records containing the details of the customer into the file corresponding to the customer text file. The account number and type of loan required by customer and other fields entered, in the data file. Records with duplicate account number fields are not allowed to be inserted. The length of the account number is checked to see whether it contains less than 10 characters.

Display

The system can then be used to display existing records of all customers. The records are displayed based on the ordering of account number maintained in the Linear-prob of indexes, which here is, an random order. Only records with references in the Hashing are

displayed. This prevents records marked as deleted (using '$') from being displayed. There is also an option to display the nodes of the Hashing level-wise.

Search

The system can then be used to search for existing records of a customer. The user is prompted for a account number, which is used as the key in searching for records in the Linear-prob of indexes. The Linear-prob is searched to obtain the desired starting byte address, which is then used to seek to the desired data record in any of the innings files. The details of the requested record, if found, are displayed, with suitable headings on the user's screen. If absent, a "record not found" message is displayed to the user.

Delete

The system can then be used to delete existing records for customer. The reference to a deleted record is removed from index while the deleted record removed in the data file. A '#' is placed in the all byte of the record, to help distinguish it from records that should be displayed. The requested record, if found, is marked for deletion, a "record deleted" message is displayed, and the reference to it is removed from the Hashing table. If absent, a "record not found" message is displayed to the user.

Modify

The system can then be used to delete existing records for customer. The reference to a deleted record is removed from index while the deleted record removed in the data file. A '#' is placed in the all byte of the record, to help distinguish it from records that should be displayed.

The requested record, if found, is marked for deletion, a "record deleted" message is displayed, and the reference to it is removed from the Hashing table. If absent, a "record not found" message is displayed to the user. If present, after deletion, the system is used to insert the modified customer record containing, possibly, a new set of details into the file corresponding to the account number.

2.4   Indexing Used

Hashing

A Hashing on the primary key is used to provide direct access to data records. Each node in the Hashing table consists of a primary key and reference pair of fields. The primary key field is the Account Number field while the reference field is the starting byte offset of the matching record in the data file.

The integer byte offset is stored in the Hashing table, and hence written to an index file, in ASCII form. On retrieval, the byte offset is converted into an integer value, before it is used to seek to a data record, as in the case of requests for a search, delete or modify operation. As records are added, nodes of the Hashing undergo splitting (on detection of overflow), merging (on detection of underflow) or redistribution (to improve storage utilization), in order to maintain a balanced Hashing structure. The data files are entry-sequenced, that is, records occur in the order they are entered into the file. The contents are loaded into their respective Hashing table, prior to use of the system, each time. Each Hashing is updated as requests for the available operations are performed. Finally, the Hashing table are written back to their files, after every insert, delete and modify operation are performed.

Chapter 3

SYSTEM DESIGN

3.1   Design of the Fields and records

In this project, we use variable length record with delimiter. In variable length record, it will take the minimum space need for that field and at the end of the record there will be a delimiter placed indicating that it is the end of the record. There are eight fields. There are:

- Accno with length 5
- Name with length 10
- Addr with length 10
- email with length 15
- phno with length 10
- occupation with length 10
- loantype with length 10

3.2   User Interface

The user interface (UI), in the industrial design field of human–computer interaction, is the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operators' decision-making process.

3.2.1   Insertion of a Record

In the insertion process, looking at the menu bar the user need to enter 1 in which records will be added into the file. When the user enters 1 they will be taken into a new screen where they need to enter certain details like the Accno, name, occupation, phone number, address and the type of loan the user needs. The means that the user is required to fill details of record of account holder with a valid input and primary key example account number should not be repeated.

- Accno which is of the type char and length from 0 to 5

- Name which is of the type char and length from 0 to 10

- Address which is of the type char and length from 0 to 10

- E-mail which is of the type char and length from 0 to 15

- Phone number which is of the type int and length from 0 to 10

- Occupation which is of the type char and length from 0 to 10

- Loan Type which is of the type char and length from 0 to 10

3.2.2 Deletion of Record

In the deletion operation the user need to enter 3. It will take the user to a new page in which it will ask the user to enter Accno. Once the Accno is entered it will try to find the record in the file. If the record is present then that record will be displayed in case the record is not there then it will display "Record not found" message on the screen. In the index file in the place of the deleted record there will be '#' placed indicating that the record has been deleted and the free space can be used to insert another new record.

3.2.3 Display of Records

In display of records, in case there is a need to view all the details of the records in the file the user can enter 2 according to the menu bar. In is option there will be a display of all the records one by one with all the details of the user like Accno, name, address, Email ID, phone number, occupation and even the lone type the user had chosen for.

3.2.4 Searching of Records

In the operation with searching of records in the file the user need to enter 2 according to the menu bar. When the user enters into the program will take the user to a new page in which the Accno to be searched need to be entered.

Once entering it, there will be a binary search for the record in the Hash file in case the record is found then there will be a display of that particular record displaying the contents of the record. If the search is unsuccessful in the sense if the record is not found, then there will be a display saying that the record is not found.

Modifying of Records

In modify operation, if the user need to modify the existing details then the

user need to enter 4 according to the menu bar. In this process the user need to enter the Accno to which modification need to be done. In case the Accno is found, then the user can make modification to the already existing details like the name, address, email ID, phone number, occupation, lone type and even the Accno.

Chapter 4

IMPLEMENTATION

Implementation is the process of defining how the system should be built, ensuring that it is operational and meets quality standards. It is a systematic and structured approach for effectively integrating software based service or component into the requirements of end users.

4.1   About C++

4.1.1   Classes and Objects

A customer file is declared as a *customer* object with the Account number, email id, occupation, address, mobile number & loan type, as its data members. An object of this class type is used to store the values entered by the user, for the fields represented by the above data members to be written to the data file.

The Hashing is declared as the class *hash*, an object of which represents a hash in the hash node. Each node contains a count of the number of entries in the node and a reference to each descendant. The maximum number of descendants is 4 while the minimum is 2.

Each node also has an array of objects, each an instance of the class type *index*. Each object of type *index* contains a ACCONO filed and an address field, which stores the ASCII representation of the starting byte address at which the corresponding data record is stored in the data file.Class objects are created and allocated memory only at runtime.

4.1.2   Dynamic Memory Allocation and Pointers

Memory is allocated for nodes of the Hashing dynamically, using the method *malloc(),* which returns a pointer (or reference), to the allocated block. Pointers are also data members of objects of type *hash node*, and, an object's pointers are used to point to the descendants of the node represented by it. File handles used to store and retrieve data from files, act as pointers. The *free()* function is used to release memory, that had once been allocated dynamically. *malloc()* and *free()* are defined in the header file *alloc.h.*

4.1.3   File Handling

Files form the core of this project and are used to provide persistent storage for user entered information on disk. The *open()* and *close()* methods, as the names suggest, are defined in the C++ Stream header file *fstream.h,* to provide mechanisms to open and close files. The *physical* file handles used to refer to the *logical* filenames, are also used to ensure files exist before use and that existing files aren't overwritten unintentionally.

The 2 types of files used are data files and index files.*open()* and *close()* are invoked on the file handle of the file to be opened/closed. *open()* takes 2 parameters- the filename and the mode of access. *close()* takes no parameters.

4.1.4   Character Arrays and Character functions

Character arrays are used to store the ACCNO fields to be written to data files and stored in a hash index object. They are also used to store the ASCII representations of the integer and integer-point fields , that are written to the data file, and, the starting byte offsets of data records, to be written to the file. Character functions are defined in the header file *ctype.h*. Some of the functions used include:

- toupper() – used to convert lowercase characters to uppercase characters.
- itoa() – to store ASCII representations of integers in character arrays
- isdigit() – to check if a character is a decimal digit (returns non-zero value) or not (returns zero value)
- isalpha() - to check if a character is an alphabet (returns non-zero value) or not (returns zero value)
- atoi() – to convert an ASCII value into an integer.
- atof() – converts an ASCII value into a floating-point number.

4.2   Pseudocode

4.2.1   Reading Customer Details Module Pseudocode

The read() function adds index objects to the Hashing if the accno entered is not a duplicate. Values are inserted into a node until it is full, after which the node is split and a new root node is created for the 2 child nodes formed. It makes calls to other recursive

and non-recursive functions. Here customer will add his/her enter certain details like the Accno, name, occupation, phone number, address and the type of loan the user needs.

```cpp
void customer :: read()
{
        cout<<"\nEnter the Account Number = ";
        gets(accno);
        cout<<"\nEnter the Customer Name = ";
        gets(name);
        cout<<"\nEnter the Customer Address = ";
        gets(Addr);
        cout<<"\nEnter the Customer Email id = ";
        gets(email);
        cout<<"\nEnter the Customer Phone Number = ";
        gets(phno);
        cout<<"\nEnter the Customer Occuptaion = ";
        gets(occupation);
        cout<<"\nEnter the Loan type = ";
        cout<<"\n\t 1.Education loan";
                cout<<"\n\t 2.Home loan";
                cout<<"\n\t 3.Car laon\n";
                int ch;
                cout<<"\n\t Enter choice:\t ";
                cin>>ch;
                if(ch==1)
                {
                        strcpy(loantype,"Education");
                }
                else if(ch==2)
                {
                        strcpy(loantype,"Home");
                }
                else
                {
```

```
                    strcpy(loantype,"Car");
                }
           pack();
            return;
      }
```

4.2.2   Display Module Pseudocode

The display function in the objects are used to retrieve and display the corresponding data record fields. Here file pointer will search for specified accno in a hash file if record found then details of specific record is displayed on console or it will displayed accno not found.

```
      void customer::display()
      {
       int i=0;
            char dummy[10];
            file.open(customerfile,ios::in|ios::out);
      cout<<"Accno"<<setw(5)<<"Name"<<setw(10)<<"Address"<<setw(10)<<
      "EmaiID"<<setw(20)<<"PhoneNo."<<setw(10)<<"occupation"<<setw(10)<<
      "loan type"<<endl;
            cout<<"----------------------------------------------------------";
            for(i=0;i<max;i++)
            {
                  file.seekp(i*recsize,ios::beg);
                  file.getline(dummy,5,'\n');
                  if(strcmp(dummy,"####")==0)
                        continue;
                  file.seekp(i*recsize,ios::beg);
                  file.getline(accno,5,'|');
                  unpack();
      cout<<accno<<setw(5)<<name<<setw(10)<<Addr<<setw(10)<<email<<
      setw(20)<<phno<<setw(10)<<occupation<<setw(10)<<loantype<<
      setw(10)<<endl;
                  cout<<"----------x---------x----------x-------x-------x-----x-------";
```

```
            }
        file.close();
        getch();
        return;
    }
```

4.2.3   Deletion Module Pseudocode

The deletion() function deletes values from nodes and balances the hash after, by merging or redistributing objects in the nodes. It makes calls to other recursive and non-recursive functions. Here file pointer will search for specified accno in a hash file if record found then details of specific record is replaced by '#' and after compeleting the operation hash file will be closed.

```
void customer :: Delete(int addr)
    {
            int j;
            strcpy(buffer,"#");
            file.open(customerfile,ios::in|ios::out);
            file.seekp(addr*recsize,ios::beg);
            for(j=0;j<recsize-2;j++)
            file<<"#";
            file<<endl;
            file.close();
    }
```

4.2.4   Retrieve Module Pseudocode

The *retrieve()* function hash, based on the values of objects in the root node. It is a recursive function.

```
    int customer::retrieve(int addr,char k[])
    {
     int found=0,i;
            char dummy[5];
            i=addr;
            file.open(customerfile,ios::in|ios::out);
```

```
            do
            {
                    file.seekg(i*recsize,ios::beg);
                    file.getline(dummy,5,'\n');
                    if(strcmp(dummy,"####")==0)
                            break;
                    file.seekg(i*recsize,ios::beg);
                    file.getline(accno,5,'|');
                    if(strcmp(accno,k)==0)
                    {
                            found=1;
                            unpack();
                            clrscr();
                            cout<<"\n\nCustomer Record found : \n";
    cout<<"Accno"<<setw(5)<<"Name"<<setw(10)<<"Address"<<
    setw(10)<<"EmailID"<<setw(20)<<"PhoneNo."<<setw(10)<<
    "occupation"<<setw(10)<<"loan type"<<endl;
    cout<<"----------------------------------------------------------------";
    cout<<accno<<setw(5)<<name<<setw(10)<<Addr<<setw(10)<<email<<
    setw(20)<<phno<<setw(10)<<occupation<<setw(10)<<loantype;
        cout<<"---------x---------x---------x------x-------x-------x--------";
                            break;
                    }
            else
            {
                    i++;
                    if(i%max==0)
                            i=0;
            }
    } while(i!=addr);
    if(found==0)
            cout<<"\n Record does not exist in Hash file\n";
            getch();
```

```
        return (found);
    }
```

4.3   Testing

Software Testing is the process used to help identify the correctness, completeness, security and quality of the developed computer software. Testing is the process of technical investigation and includes the process of executing a program or application with the intent of finding errors.

4.3.1   Unit Testing

Unit Testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

- Accno. input is checked to see if it is of 5 digit or not.
- Name input is checked to see if it is of 10 character or not.
- Address input is checked to see if it is of 10 character or not.
- Email ID input is checked to see if it is of 15 character or not.
- Loan Type input is checked to see if it is of 10 character or not.
- Phone no. input is checked to see if it is of 10 character or not.

Table 4.1 Unit test case for Input Check

CaseID	Description	Input Data	Expected Output	Actual Output	Status
1.	Enter valid accno.	12345	Press any key for next value	Press any key for next value	PASS
2.	Enter Invalid accno	123456	Accno not in range	Accno not in range	PASS
3.	Enter Name	Satyam	Press any key for next value	Press any key for next value	PASS
4.	Enter Address	Muzzfarnagar	Press any key for next value	Press any key for next value	PASS

5.	Enter email	sm@gmail.co	Press any key for next value	Press any key for next value	PASS
6.	Enter loantype	Eduction	Press any key for next value	Press any key for next value	PASS
7.	Enter Phone number	9758519066	Press any key for next value	Press any key for next value	PASS

4.3.2 Integration Testing

The insertion function is checked to see if a duplicate record is attempted to be inserted and that the data files and index files are correctly updated with the appropriate records. It is also checked to see if validation of input values is performed correctly. Accno. Is a primary key, so repetition of accno. Is not allowed. If accno. Is repeated then appropriate message should be displayed. Other enteries such as name, address, email, loantype, phone number can be repeated. Every entry should have different account number.

Table 4.2 Integrate test case for Insertion.

Case ID :	1
Name of test:	Integration Check test
Item / Feature being tested:	Insertion module
Sample Input:	ACCNO='56789'. Upon press of ENTER key.
Expected output:	Accno. and name inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message displayed.
Actual output:	Accno. and name inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message is displayed.
Status	SS

2. The deletion function is checked to see if validation of input values is performed

correctly, the correct results are returned based on whether a file contains records or not and whether desired record is present in a file or not, only existing matching records are deleted and that the result of the deletion is reflected in the data and index files.

Table 4.3 Integration test case for Deletion module

Case ID :	2
Name of test:	Check test
Item / Feature being tested:	Deletion module
Sample Input:	ACCNO='56789'. Upon press of ENTER key.
Expected output:	"Record Deleted" message followed by "Press any key to go back to Menu" message displayed.
Actual output:	"Record Deleted" message followed by "Press any key to go back to Menu" message is displayed.
Status	PASS

3. The search function is checked to see if validation of input values is performed correctly, correct results are returned based on whether a file contains records or not and whether desired record is present in a file or not.

Table 4.4 Integration test case for Retrieve module

Case ID :	3
Name of test:	Check test
Item / Feature being tested:	Retrieve module
Sample Input:	ACCNO='56789'. Upon press of ENTER key.
Expected output:	"Record found" message followed by details of the record and a "Press any key to go back to Menu" message displayed.
Actual output:	"Record found" message followed by details of the record and a "Press any key to go back to Menu" message is displayed.

Status	PASS

4.3.3 System Testing

System testing is defined as testing of a complete and fully integrated software product. This testing falls in black –box testing where in knowledge of the inner design of the code is not a pre-requisite and is done by the testing team. System testing is done after integration testing is complete. System testing should test functional and non-functional requirements of the software.

Table4.5 System Testing for Customer record details

Case id	Description	Input data	Expected output	Actual output	Status
1	To display the records	Display records for valid Accno='123'(present)andinvalid Accno ='123456'(notpresent)	Record is displayed for valid and record not found for invalid accno	Record is displayed for valid accno and record not found for invalid accno	Pass
2	To search the records	Display records for valid Accno='123'(present)andinvalid Accno ='123456'(notpresent)	Record is displayed for valid and record not found for invalid	Record is displayed for valid and record not found for invalid	Pass
3	To delete the records	Delete records for valid accno = '123'(present) and invalid accno='123456'(not present)	Record is deleted for valid and record not found for invalid	Record is deleted for valid and record not found for invalid	Pass
4	To update the records	Update records for valid accno. = '123'(present) and	Record is Updated for valid and record	Record is Updated for valid and	Pass

Case id	Description	Input data	Expected output	Actual output	Status
		invalid accno = '123456'(notpresent)	not found for invalid	record not found for invalid	

4.3.4   Acceptance Testing

Acceptance Testing is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery. It is performed by:

- Internal Acceptance Testing (Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.
- External Acceptance Testing is performed by people who are not employees of the organization that developed the software.
    - Customer Acceptance Testing is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software. [This is in the case of the software not being owned by the organization that developed it.]
    - User Acceptance Testing (Also known as Beta Testing) is performed by the end users of the software. They can be the customers themselves or the customers' customers.

Test Cases for Acceptance testing can be on three possible module – insertion module, deletion module, updation module.

Table4.6 Acceptance Testing for Customer record details

Case ID	Description	Input Data	Expected Output	Actual Output	Status
1.	Insertion Module	Inserting valid data	Record added successfully	Record added successfully	Pass
2.	Deletion Module	Enter valid accno.	Record deleted successfully	Record deleted successfully	Pass
3.	Updatation Module	Enter valid accno.	Record updated successfully	Record updated successfully	Pass

4.4   Discussion of Results

All the menu options provided in the application and its operations have been presented in as screen shots from Figure 4.1 to 4.10

4.4.1   Menu Options

Here we can choose options to perform operation various like adding a new record, searching and displaying a record, modifying existing record, delete existing record and display all customer records.
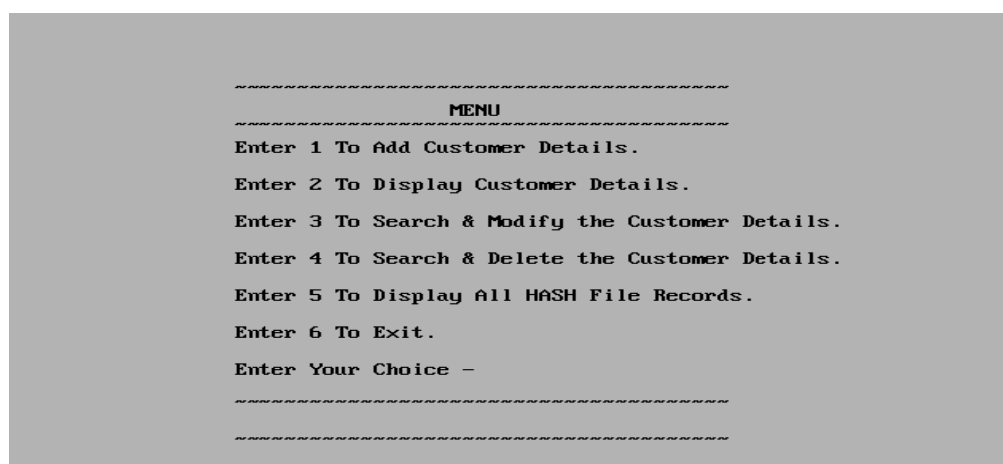
Figure 4.1 User Menu Screen

4.4.2   Adding Customer Record

Here we enter user account number, name, address, email id, contact number, his/her occupation and type of loan user applying for. Details of customer are stored in a hash file.


```
Enter Customer Details:

Enter the Account Number = 9826

Enter the Customer Name = Ritik

Enter the Customer Address = Noida

Enter the Customer Email id = ra@gmail

Enter the Customer Phone Number = 7406597267

Enter the Customer Occuptaion = Android

Enter the Loan type =
        1.Education loan
        2.Home loan
        3.Car laon

        Enter choice:   2
Done...
```

Figure 4.2 Reading of a Customer Record.
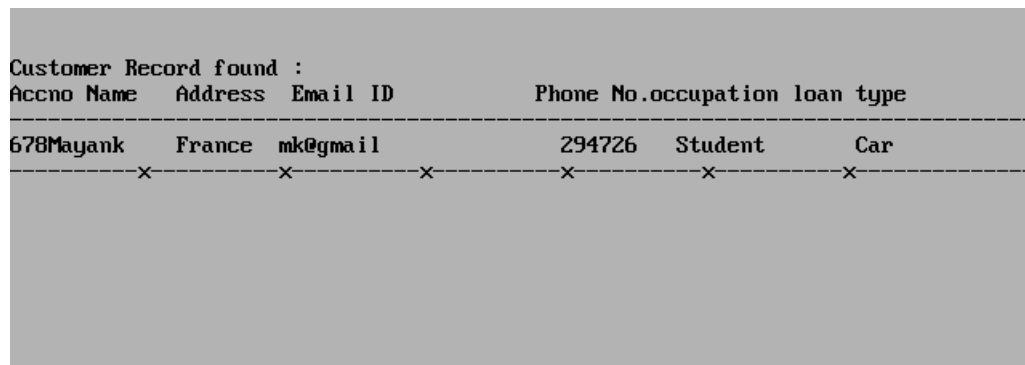

```
341|Suryansh|UA|sg@gmail|2421341|Studnet|Education|#######################
234|Satyam|Kengeri|ss@gmail|9826255|Student|Education|###################

941|Chirayu|CAlifor|CC@gmail|2421341|Bussiness|Education|#################

098|Dhruv|Gwalior|ds@gmail|4582015|Saloon|Home|######################

678|Mayank|France|mk@gmail|294726|Student|Car|#######################
679|Vaibhav|Jayanagar|vj@gmail|84627143|Engineer|Home|################
789|Sanjeev|Denmark|ss@gmail|8740281|Developer|Home|################
9826|Ritik|Noida|ra@gmail|7406597267|Android|Home| NEW RECORD INSERTED ######
```

Figure 4.3 Customer Record in a File

4.4.3 Displaying Customer Record

User need to enter account number need to be displayed.

```
Customer Record found :
Accno Name    Address  Email ID            Phone No.occupation loan type
-------------------------------------------------------------------------
678Mayank     France   mk@gmail            294726    Student      Car
---------x---------x---------x---------x----------x---------x------------
```

Figure 4.4 Displaying Customer Record.

4.4.4 Deleting Customer Record

User need to enter account number need to be deleted. Record is replaced with "#" and all details are removed from text file.

```
                    Enter Account Number9826
Customer with 9826 is deleted Successfully..
```

Figure 4.5 Deleting Customer Record.

4.4.5 Modifying Customer Record

Here we enter user account number, name, address, email id, contact number, his/her occupation and type of loan user applying for. Details of customer are updated in existing record in a hash file.

Figure 4.6 Modifying Customer Record.

Figure 4.7 Modified Customer Record in File

4.4.6 Displaying all Records

```
Accno Name     Address   Email ID        Phone No.occupation loan type
------------------------------------------------------------------------
341Suryansh       UA  sg@gmail             2421341    Studnet Education
--------------x---------x---------x---------x---------x---------x--------
234Satyam   Kengeri  ss@gmail             9826255    Student Education
--------------x---------x---------x---------x---------x---------x--------
941Chirayu    CAlifor  CC@gmail            2421341 Bussiness
--------------x---------x---------x---------x---------x---------x--------
098Dhruv   Gwalior  ds@gmail             4582015    Saloon      Home
--------------x---------x---------x---------x---------x---------x--------
678Mayank     France  mk@gmail             294726    Student      Car
--------------x---------x---------x---------x---------x---------x--------
679Vivek    Jayang  vm@gmail             2421341    Student     Home
--------------x---------x---------x---------x---------x---------x--------
789Sanjeev   Denmark  ss@gmail             8740281 Developer
--------------x---------x---------x---------x---------x---------x--------
```

Figure 4.8 Displaying all Customer Records

4.4.7 Hash File

```
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
341|Suryansh|UA|sg@gmail|2421341|Studnet|Education|#############################
234|Satyam|Kengeri|ss@gmail|9826255|Student|Education|###########################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
941|Chirayu|CAlifor|CC@gmail|2421341|Bussiness|Education|#########################
#######################################################################################
#######################################################################################
098|Dhruv|Gwalior|ds@gmail|4582015|Saloon|Home|#################################
#######################################################################################
#######################################################################################
#######################################################################################
678|Mayank|France|mk@gmail|294726|Student|Car|##################################
679|Vivek|Jayang|vm@gmail|2421341|Student|Home|#################################
#######################################################################################
789|Sanjeev|Denmark|ss@gmail|8740281|Developer|Home|############################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
#######################################################################################
```

Figure 4.9 Hash File Conataining all Customer Records

4.4.8   Data File

```
234|Satyam|Kengeri|ss@gmail|9826255|Student|Education|
789|Sanjeev|Denmark|ss@gmail|8740281|Developer|Home|
678|Mayank|France|mk@gmail|294726|Student|Car|
679|Vaibhav|Jayanagar|vj@gmail|84627143|Engineer|Home|
098|Dhruv|Gwalior|ds@gmail|4582015|Saloon|Home|
341|Suryansh|UA|sg@gmail|2421341|Studnet|Education|
941|Chirayu|CAlifor|CC@gmail|2421341|Bussiness|Education|
679|Vivek|Jayang|vm@gmail|2421341|Student|Home||
```

Figure 4.10 Data File Conataining all Customer Records

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

This mini project facilitated storage of loan data into the file. various operation like insert, delete, search and modify operations were performed on data file containing relevant loan data.

Later, in order to make search faster, hash indexing technique is built on data file. Hash indexing uses fold and add algorithm and perform hashing of primary keys. This value used as address of a record to store and retrieve whenever is nessacary. Further, indexing like B-tree, B+ tree, can be used to make searching more efficient.

REFERENCES

[1] Raghu Ramkrishna and Johannes Gehrke, Data Managenent System,Mc GRAWHILL

[2] Ramez elmasri and Shamkant B.Navathe Functionas of Database System Pearson, 7thEdition.

[3] Michael J. Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object Oriented Approach with C++, 3rd Edition, Pearson Education, 1998.

[4] www.stackoverflow.com

[5] www.tutorialpoint.com

[6] www.wikepedia.com

[7] www.softwaretestingfundamentals.com