VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

Store Billing System using Hashing Indexing Technique

Submitted in partial fulfillment of the requirements as a part of the File Structures Lab of VI semester for the award of degree of ***Bachelor of Engineering*** *in* ***Information Science and Engineering****, Visvesvaraya Technological University, Belagavi*

Submitted by

D NANDA KISHORE
1RN16IS029

NISHITH A
1RN16IS059

Faculty Incharge
Mr. R Rajkumar
Assistant Professor

Coordinator
Mr. Santosh Kumar
Assistant Professor

Department of Information Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, R R Nagar Post
Bengaluru – 560 098

2018 – 2019

RNS Institute of Technology

Channasandra, Dr.Vishnuvardhan Road, RR Nagar Post,

Bengaluru – 560 098

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled **STORE BILLING SYSTEM USING HASHING INDEXING TECHNIQUE** has been successfully completed by **D NANDA KISHORE** bearing USN **1RN16IS029** and **NISHITH A** bearing USN **1RN16IS059**, presently VI semester students of **RNS Institute of Technology** in partial fulfillment of the requirements as a part of the **FILE STRUCTURES** Laboratory for the award of the degree of ***Bachelor of Engineering in Information Science and Engineering*** under **Visvesvaraya Technological University, Belagavi** during academic year **2018 – 2019**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements as a part of File structures Laboratory for the said degree.

Mr. R Rajkumar	**Mr. Santosh Kumar**	**Dr. M V Sudhamani**
Faculty Incharge	Coordinator	Professor and HoD
Assistant Professor	Assistant Professor	

External Viva

Name of the Examiners	**Signature with date**
1. ____________________	____________________
2. ____________________	____________________

ABSTRACT

The Store Billing System provides a user friendly, interactive Menu Driven Interface (MDI) based on local file system. All data is stored in files on disks. The program uses file handling to access the files.

This mini project is a traditional Store Billing System with some added functionality. This system is built for fast data processing and bill generation for Store customers.

The billing database is a vast collection of product name, price and other product specific data. When a product is searched from the file , its price is added to the bill based on the product quantity. The Store billing system is built to help staff members to calculate and display the bill and serve the customer in a faster and efficient manner.

The project work consists of an effective and easy MDI to help the staff in easy bill calculation and providing an efficient customer service. This mini project preserves the identity of fields by separating them with delimiters. 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. Pipe Symbol ('|') has been used as the record delimiter.

This mini project uses Hash Indexing Technique to look up the position of a record in the file using the primary key. A Hashing Index uses the primary key to provide direct access to data records.

ACKNOWLEDGMENT

The fulfillment 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.

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

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

We would like to express our thanks to our Principal **Dr. M K Venkatesh** for his support and inspired me towards the attainment of knowledge.

We 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.

We 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 his valuable guidance, constructive comments and continuous encouragement throughout the Project work.

We would like to express our profound and cordial gratitude to our guide **Mr R RajKumar**, Assistant Professor, Department of Information Science and Engineering for his valuable guidance in preparing Project report.

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

NISHITH A USN: 1RN16IS059

D NANDA KISHORE USN: 1RN16IS029

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

STORE BILLING SYSTEM using hashing technique is used to store the product details such as the product id, name of the product and its price. It is used to manage the billing process in large departmental stores. The administrator can create, delete, search and modify products. The staff can search for the product needed by the customer and produce the invoice for the customer.

1.1 Introduction to File Structure

A file structure is a combination of representations for data in files and of operations for accessing the data. A file structure allows applications to read, write, and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order. An improvement in file structure design may make an application hundreds of times faster. The details of the representation of the data and the implementation of the operations determine the efficiency of the file structure for particular applications.

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.

In the early 1960's, the idea of applying tree structures emerged. But trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record. In 1963, researchers developed an elegant, self-adjusting binary tree structure, called AVL tree, for data in memory. The problem was that, even with a balanced binary tree, dozens of accesses were required to find a record in even moderate-sized files. A method was needed to keep a tree balanced when each node of thee tree was not a single record, as in a binary tree, but a file block containing dozens, perhaps even hundreds, of records .

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 of adding structure to files to maintain the identity of fields are:

- Force the fields into a predictable length.
- 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

The end of the field can be stored by counting 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. The fields are referred as length-based.

Method 3: Separate the Fields with Delimiters

The identity of fields can be preserved by separating them with delimiters. It is needed to 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.

```
a)  Ames     Mary    123 Maple    Stillwater    OK74075
    Mason    Alan    90 Eastgate  Ada           OK74820

b)  04Ames04Mary09123 Maple10Stillwater02OK0574075
    05Mason04Alan1190 Eastgate03Ada02OK0574820

c)  Ames|Mary|123 Maple|Stillwater|OK74075|
    Mason|Alan|90 Eastgate|Ada|OK|74820|

d)  Last=Ames|first=Mary|address=123 Maple|city=Stillwater|state=OK|zip=74075|
```

Figure 1.1 Four methods for field structures

Record Structures

The five most often used methods for organizing records are:

- Require the records to be predictable number of bytes in length.
- Require the records to be predictable number of fields in length.
- 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. In the figure below, we have 6 contiguous fields and we can recognize fields simply by counting the fields modulo 6.

Method 3: Begin Each Record with a Length Indicator

Communication of 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

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, use a # character as the record delimiter.

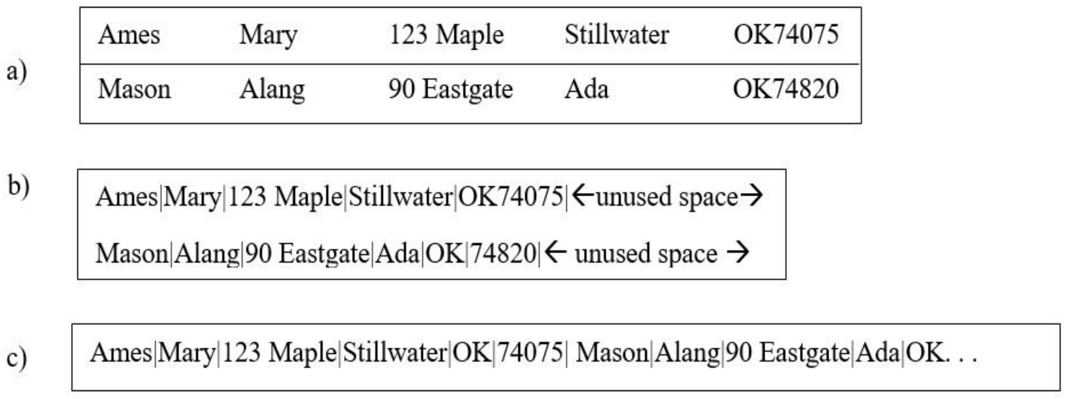

a)
Ames Mary 123 Maple Stillwater OK74075
Mason Alang 90 Eastgate Ada OK74820
b)
Ames|Mary|123 Maple|Stillwater|OK74075|←unused space→
Mason|Alang|90 Eastgate|Ada|OK|74820|← unused space →
c)
Ames|Mary|123 Maple|Stillwater|OK|74075| Mason|Alang|90 Eastgate|Ada|OK. . .

Figure 1.2 Making Records Predictable number of Bytes and Fields

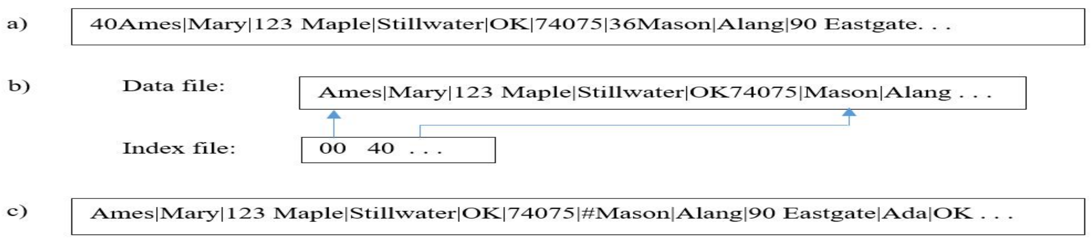

a)
40Ames|Mary|123 Maple|Stillwater|OK|74075|36Mason|Alang|90 Eastgate. . .
b)
Data file:
Ames|Mary|123 Maple|Stillwater|OK74075|Mason|Alang . . .
Index file:
00 40 . . .
c)
Ames|Mary|123 Maple|Stillwater|OK|74075|#Mason|Alang|90 Eastgate|Ada|OK . . .

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. 1 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) is 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, non-volatile capacity, is the driving force behind file structure design. Good file structure design will give us access to all the capacity without making this applications spend a lot of time waiting for the disk.

Chapter 2

SYSTEM ANALYSIS

2.1 Analysis of Application

Store Billing System can be used by the administrator who can create, delete, search and modify the product details. The staff can used this system to select the products needed by the customer and produce the invoice for them.

2.2 Structure used to Store the Fields and Records Storing Fields

Fixing the Length of Fields:

In the Store Billing System, the PID 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. Simple arithmetic is sufficient to recover data from the original fields.

Separating the Fields with Delimiters:

We preserve the identity of fields by separating them with delimiters. We have chosen the vertical bar character, as the delimiter here.

Storing Records

Making Records a Predictable Number of Fields:

In this system, 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.

2.3 Operations Performed on a File

▪ **Insertion**

The system is used to add product record containing Pid, product name and the price of the product into the file. Records with duplicate Pid fields are not allowed to be inserted. The length of the Pid is checked to see whether it contains only 5 characters.

▪**Retrieve**

The system can then be used to search for existing records of Product. Retrieves the set of record for the given search Pid. The Hashing is searched to obtain the desired starting byte address, which is then used to seek to the desired data record in any of the 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 from product file. The reference to a deleted record is removed from index while the deleted record persists in the data file. 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 file. If absent, a "record not found" message is displayed to the user. The deleted record space is filled with " # " symbol.

▪ **Modify**

The system can be used to insert the modified product record in the file.Modify the set of records based on single Pid.The requested record, if found, is marked for modification, and stores the correct modified record in the file and display the message as "record updated successfully", and the reference to it is removed from the Hashing file. If absent, a "record not found" message is displayed to the user.

2.4 Indexing Used

Hashing

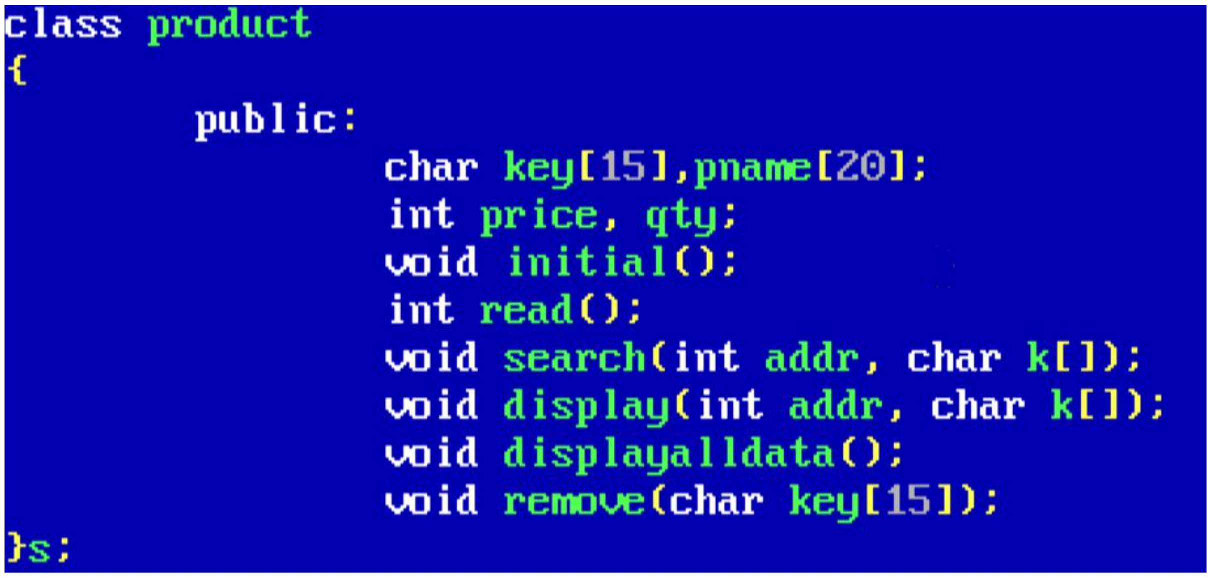
A hash function is like a black box that produces an address every time you drop in a key. It is a function h (K)that transforms a key K into an address. The address generated appear to be random hence hashing is also referred as randomizing. The resulting address is used as the basis for storing and retrieving record. When this occurs, it is called collision and some means must be found to deal with it. When two different keys may be transformed to the same address so two records may be sent to same place in the file. When the key in our project is the scientist identification number (Pid) for every insert, delete and modify operation.

Chapter 3

SYSTEM DESIGN

3.1 Design of the Fields and Records

The Pid is declared as a character array that can hold a maximum of 15 characters. Checks are done to ensure the Pid if of exactly 15 characters during input. The pname is declared as a character array that can hold a maximum of 20 characters. Price and qty are of the integer type. Hence, a typical product data file record can have up to 110 bytes of information to be stored. This includes the 18 bytes taken up by the field delimiters ('|'). The class declaration of a typical scientist file record is as shown in Figure 3.1:

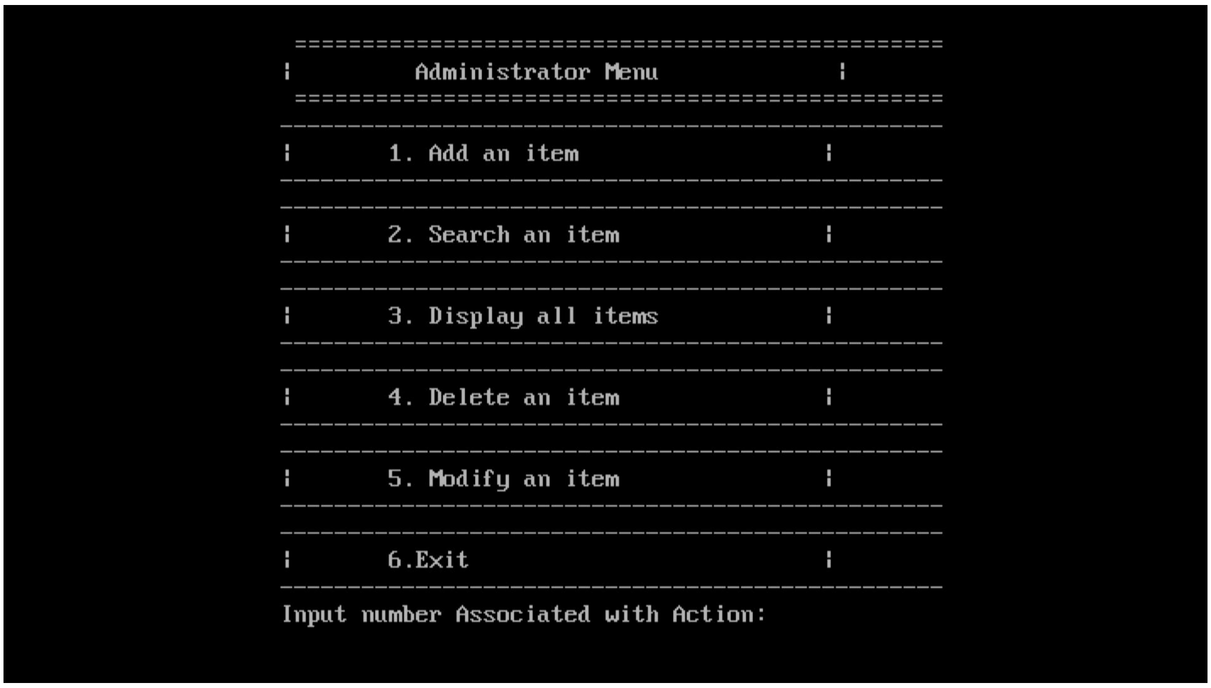


```
class product
{
        public:
                char key[15],pname[20];
                int price, qty;
                void initial();
                int read();
                void search(int addr, char k[]);
                void display(int addr, char k[]);
                void displayalldata();
                void remove(char key[15]);
}s;
```

Figure 3.1 Class Product

3.2 User Interface

The User Interface or UI refers to the interface between the application and the user. Here, the UI is menu-driven, that is, a list of options (menu) is displayed to the user and the user is prompted to enter an integer corresponding to the choice that represents the desired operation to be performed.



```
 ================================================
|          Administrator Menu                    |
 ================================================
-------------------------------------------------
|        1. Add an item                        |
-------------------------------------------------
-------------------------------------------------
|        2. Search an item                     |
-------------------------------------------------
-------------------------------------------------
|        3. Display all items                  |
-------------------------------------------------
-------------------------------------------------
|        4. Delete an item                     |
-------------------------------------------------
-------------------------------------------------
|        5. Modify an item                     |
-------------------------------------------------
-------------------------------------------------
|        6.Exit                                |
-------------------------------------------------
Input number Associated with Action:
```

Figure 3.2 Menu Screen

3.2.1 Insertion of a Record

If the operation desired is Insertion, the user is required to enter 1 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the Pid, product name and product price which will be inserted into the file. The user is prompted for each input until the value entered meets the required criterion for each value.

3.2.2 Display all Record

If the operation desired is Display of Records, the user is required to enter 3 as his/her choice, from the menu displayed, after which a new screen is displayed. If there are no records in any file, it display a message as “no records found”. For the product file all the details of each record within the file, with suitable headings, is displayed.

3.2.3 Deletion of a Record

If the operation desired is Deletion, the user is required to enter 4 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the key whose record from the file is to be deleted. If the record is not found then “record not found” message is displayed. If the record is found, a “record deleted” message is displayed. The deleted record space is filled with “#” symbol.

3.2.4 Searching of a Record

If the operation desired is Search, the user is required to enter 2 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the key whose record is to be searched for in the file. If the record is not found, a “record not found” message is displayed. If one is found, the details of the record, with suitable headings, are displayed.

3.2.5 Modifying of a Record

If the operation desired is Modify, the user is required to enter 5 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the key whose record has to be modified in the file. The Modify operation is implemented as a deletion followed by an insertion. If the record is not found in the file, a “no records to delete” message is displayed. If the record is found, then the user is asked to enter the new details of the product and the respective changes will be done to the file.

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 a software based service or component into the requirements of end users.

4.1 About C++

C++ is a general purpose programming language. It has imperative, object –oriented and generic programming features, while also providing facilities for low level memory manipulation

4.1.1 Classes and Objects

The product file is declared as class with the Pid, pname, price and qty as its data members and Initial (), read (), search (), display (), displayalldata (), remove () as class methods. 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. Each an instance of the class type index. The data stored in file according to hash() which returns the 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 s 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.

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.

4.2 Pseudo code

4.2.1 Insertion Module Pseudo code

The read method is used to add new product details to the file.Values are inserted into a hash table until it is full.

Function to read the data into buffer

```
{       char a[10], b[5];
        cout<<"Enter product key:";          cin>>key;
        for(int j=0; j<indsize; j++    )
        {
                if(strcmp(id[j].ikey,key)==0)
                {
                        cout<<"Key already exists"<<endl;
                        getch();
                        clrscr();
                        return -1;
                }
        }
        cout<<"Enter the Name:";             cin>>pname;
        cout<<"Enter Price:";                cin>>price;
        strcpy(buffer, key);                 strcat(buffer, "|");
        strcat(buffer,pname);                strcat(buffer,"|");
        itoa(price,a,10);
        strcat(buffer,a);                    strcat(buffer,"|");
        return 1;
}
```

Function to store the data

```
{
        int flag=0,i;
        file.open(shopfile,ios::in|ios::out);
        file.seekg(addr*recsize,ios::beg);
        file.getline(dummy,5,'\n');
        if(strcmp(dummy,"####")==0)
        {       file.seekp(addr*recsize,ios::beg);
                file<<buffer;
         flag=1;
}
        else
        {       for(i=addr+1; i!=addr; i++)
                {       if(i%max==0)
                                i=0;
                        file.seekg(i*recsize,ios::beg);
                        file.getline(dummy,5,'\n');
                        if(strcmp(dummy,"####")==0)
                        {
                                cout<<"\n Collision has occored\n";
                                file.seekp(i*recsize,ios::beg);
                                file<<buffer;
                                flag=1;
                                break;
                        }
                }
        }
        if(i==addr && (!flag))
                        cout<<"hash File is full, Record cannot be inserted\n";
                file.close();
                return;
}
```

4.2.2 Display Module Pseudo code

The displayalldata() method fetch all the data present in the file and display into console. Check whether the hash table is filled with ‘ # ’ if not then display the contents present in the file.

```
Function to display all data
{
        cout<<"\n Display all items:\n";
        cout<<setw(15)<<"key"<<setw(20)<<"pname"<<setw(10)<<"price"<<endl;
        for(int j=0; j<indsize; j++    )
        {
                addr=hash(id[j].ikey);
                s.display(addr,id[j].ikey);
        }do
        {
                cout<<"\nPress 1 to exit:";
                cin>>a;
                if(a==1)
                break;
        }while(1);
}
```

4.2.3 Deletion Module Pseudo code

The remove() method deletes the record from the file ,the user is prompted for the Pid, whose matching record is to be deleted. The deleted record space is filled with “#” symbol.

```
Function to remove a record with key[] as primary key
{
        int pos=-1,flag,addr,found,j;
        char k[20],dummy[150],a[10];
        ind1.intl();
        for(int i=0;i<=indsize;i++)
        {
                if(strcmp(id[i].ikey,key)==0)
                {
                        pos=i;
                        flag=1;
                        break;
                }
        }
        if(flag==1)
        {
        for( i=pos;i<indsize; i++)
                id[i]=id[i+1];
                indsize--;
        ind1.readi();
        }else
        {
                return;
        }
        addr=hash(key);
        i=addr;
        file.open(shopfile,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(k,15,'|');
         if(strcmp(key,k)==0)
         {
                found=1;
                file.getline(pname,20,'|');
                file.getline(a,10,'|');
                file.seekp(i*recsize,ios::beg);
                for(j=0;j<recsize-2;j++)
                        file<<"#";
                file<<endl;
                break;
        }
        else
        {
                i++;
                if(i%max==0)
                i=0;
        }
 } while(i!=addr);
 if(found==0)
 cout<<"\n\t\t\trecord does not exist in hash file\n";
 getch();
 file.close();
 return;
}
```

4.2.4 Search Module Pseudocode

The search() function searches the file for the record of the Pid entered by the user and displays the corresponding record details.

```
Function to search a record by passing address and key
{
        int found=0,i;
        char dummy[10],a[10];
        i=addr;
        file.open(shopfile,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(key,15,'|');
                if(strcmp(key,k)==0)
                {
                        found=1;
                        file.getline(pname,20,'|');
                        file.getline(a,10,'|');
                     price= atoi(a);
                        cout<<"key="<<key<<"\nname="<<pname<<"\nprice="<<price;
                                break;
                }
                else
                {       i++;
                        if(i%max==0)
                                i=0;
                }
        }while(i!=addr);
        if(found==0)
                cout<<"\n Record Does not exists in hash file\n";
        file.close();
        return;
}
```

4.2.5 Modify Module Pseudo code

The modify() method operation is implemented as a call to the hash and remove functions followed by a call to the insertion function.

```
                Modify the record

                {

                        cout<<"Enter ket of element to be modified:\n";
                        cin>>skey;
                        s.remove(skey);
                        cout<<endl<<"\nenter modified item\n " <<endl;
                           t= s.read();
                          if(t==1)
                {
                    addr=hash(s.key);
                    store(addr);
                    ind1.intl();
                    strcpy(id[indsize].ikey,s.key);
                    cout<<id[indsize].ikey<<endl;
                    indsize++;
                    ind1.readi();
                }
        }
```

4.2.7 Indexing Pseudo code

The hash indexing is written to the index file after every insertion, deletion, and modification operation, using the hash() function to keep index file up-to-date and consistent and returns the address to store the data in to the file.

```
Hash Function for key[]
{
        int i=0,sum=0;
        while(key[i]!='\0')
        {
                sum=sum+key[i]-48;
                i++;
        }
        return sum % max;
}
```

4.3 Testing

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. Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test.

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.

1. Pid input it checked to see if it is in alphanumeric form:

Table 4.1 Unit test case for Pid Input Check

Sl No. of test case:	1
Name of test:	Check test
Item / Feature being tested:	Input for Pid field
Sample Input:	Pid='adm01'. Upon press of ENTER key.
Expected output:	Prompt for Pid with 'Enter Pid:' with "Pid accepts only numeric" message.
Actual output:	'Enter Pid:' with "Pid accepts only numeric" message displayed.
Remarks:	Test succeeded

2. Product name input is checked to see if it contains only characters:

Table 4.2 Unit test case for pname Input Check

Sl No. of test case:	2
Name of test:	Check test
Item / Feature being tested:	Input for product name field
Sample Input:	Product name='Cookie'. Upon press of ENTER key.
Expected output:	Prompt for depot name with 'Enter product name:' with "Product name accepts only characters" message.
Actual output:	'Enter Product name:' with "product name accepts only characters" message displayed.
Remarks:	Test succeeded

3. Price input it checked to see if it contains only numerals:

Table 4.3 Unit test case for price Input Check

Sl No. of test case:	3
Name of test:	Check test
Item / Feature being tested:	Input for price field

Sample Input:	price='1000'. Upon press of ENTER key.
Expected output:	Prompt for destination with 'Enter price:' with "price accepts only numerals" message.
Actual output:	'Enter price:' with "price accepts only characters" message displayed.
Remarks:	Test succeeded

4.3.2 Integration Testing

Integration Testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

1. The insertion function is checked to see if duplicate record is attempted to be inserted and that the data files and secondary storage file are correctly updated with the appropriate records .It is also checked to see if validation of input values is performed correctly.

Table 4.4 Integration test case for Insertion module

Sl No. of test case:	1
Name of test:	Check test
Feature being tested:	Read module
Sample Input:	Pid=001'. Upon press of ENTER key.
Expected output:	Pid and inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message displayed.
Actual output:	Pid and other fields accepted, followed by 'Record store' message and a 'Press any key to go back to Menu' message is displayed.
Remarks:	Test succeeded

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 4.5 Integration test case for Deletion module

Sl No. of test case:	2
Name of test:	Check test
Item / Feature being tested:	Delete module
Sample Input:	Pid='005'. Upon press of ENTER key.
Expected output:	"Record not found" message followed by "Press any key to go back to Menu" message displayed.
Actual output:	"Record not found" message followed by "Press any key to go back to Menu" message is displayed.
Remarks:	Test succeeded

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 4.6 Integration test case for Search module

Sl No. of test case:	3
Name of test:	Check test
Item / Feature being tested:	Search module
Sample Input:	Pid='001'. Upon press of ENTER key.
Expected output:	"Record found" message followed by record data which belong to Pid.
Actual output:	"Record found" message followed by record data which belong to Pid..
Remarks:	Test succeeded

4. The modify function is checked to see if 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 records are deleted, the deletion followed by insertion happens in the proper way, validation of input values is performed correctly, and that the result of the modification is reflected in the data and index files.

Table 4.7 Integration test case for Modify module

Sl No. of test case:	4
Name of test:	Check test
Item / Feature being tested:	Modify module
Sample Input:	Sid='001'. Upon press of ENTER key.
Expected output:	"Record found" message followed by prompts to enter the new data.
Actual output:	"Record found" message followed by prompts to enter the new data.

4.3.3 System Testing

System Testing is a level of the software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. The application is run to check if all the modules (functions) can be executed concurrently, if each return correct results of the operations performed by them, and if the data and index files are left in consistent states by each module.

Table 4.8 System test case for Product details

Sl No. of test case:	1
Name of test:	Check test
Item / Feature being tested:	Product Details
Sample Input:	Choices entered in the order: 1 1 3 1 2 4 3 5 4 3

Expected output:	Screens displayed (except menu screen) in order for: Admin page Insertion of record Display of all records in the file Insertion of record Searching of record Deletion of a record. Display of each records in file Modifying of record Remove of a record Display of each records in data file
Actual output:	Screens are displayed in order
Remarks:	Test succeeded

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, modify module.

Table4.9 Acceptance Testing for product record details

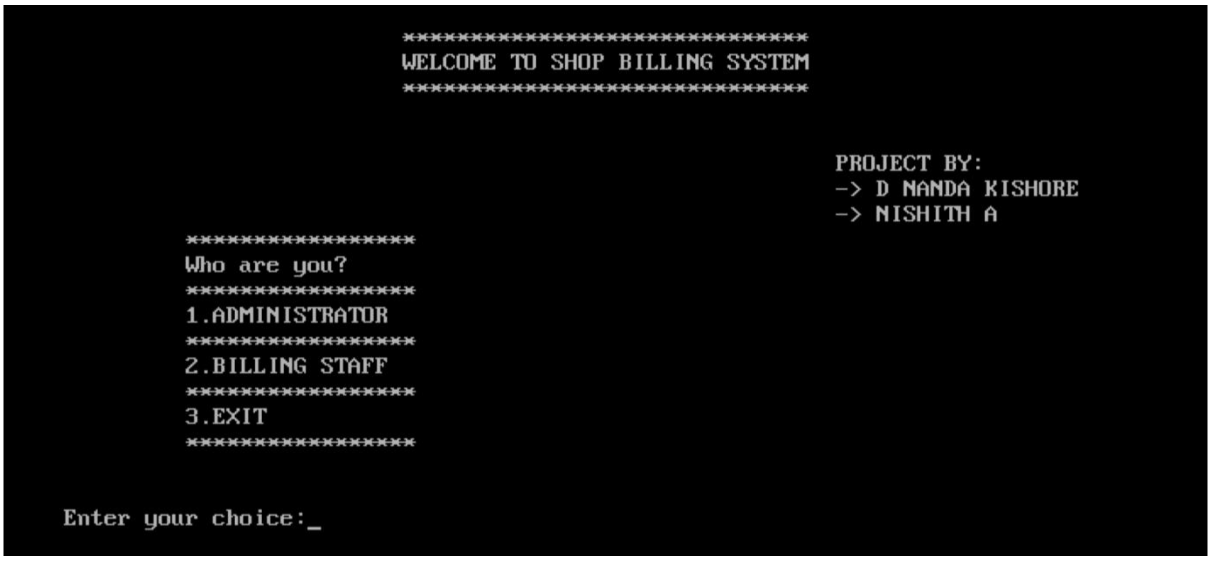
Case ID	Description	Input Data	Expected Output	Actual Output	Status
1.	Insertion Module	Inserting valid data	Record added successfully	Record added successfully	Pass
2.	Deletion Module	Enter valid Pid.	Record deleted successfully	Record deleted successfully	Pass
3.	Modify Module	Enter valid Pid	Record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

4.4.1 Starting Menu Options

This is the opening screen.

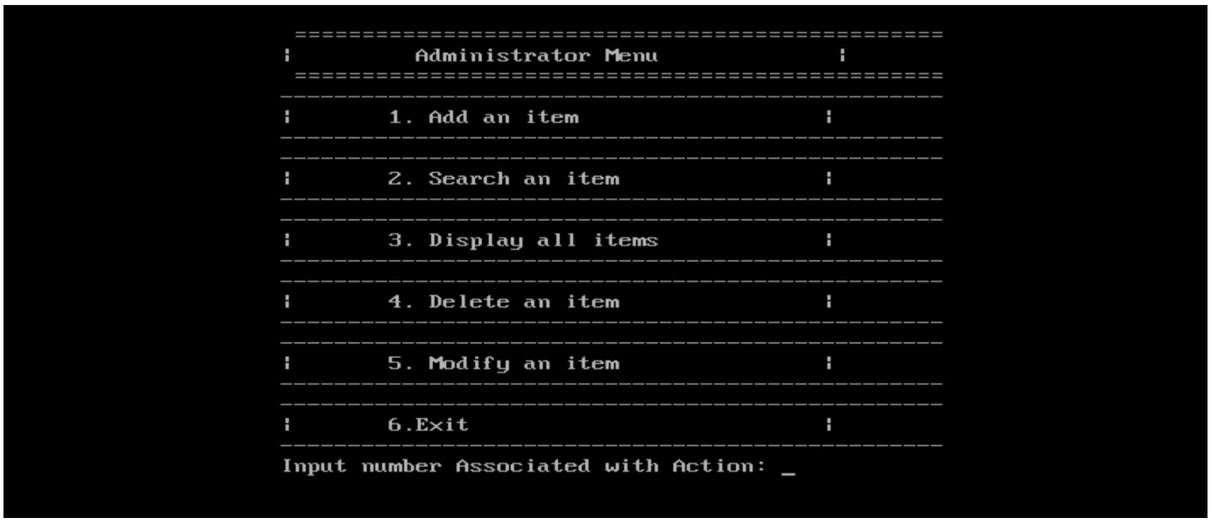


```
                              ******************************
                              WELCOME TO SHOP BILLING SYSTEM
                              ******************************

                                                                PROJECT BY:
                                                                -> D NANDA KISHORE
                                                                -> NISHITH A
              *****************
              Who are you?
              *****************
              1.ADMINISTRATOR
              *****************
              2.BILLING STAFF
              *****************
              3.EXIT
              *****************

     Enter your choice:_
```

Figure 4.1 Starting Menu Screen.

4.4.2 Admin menu

This menu pops up when you login in as the admin.

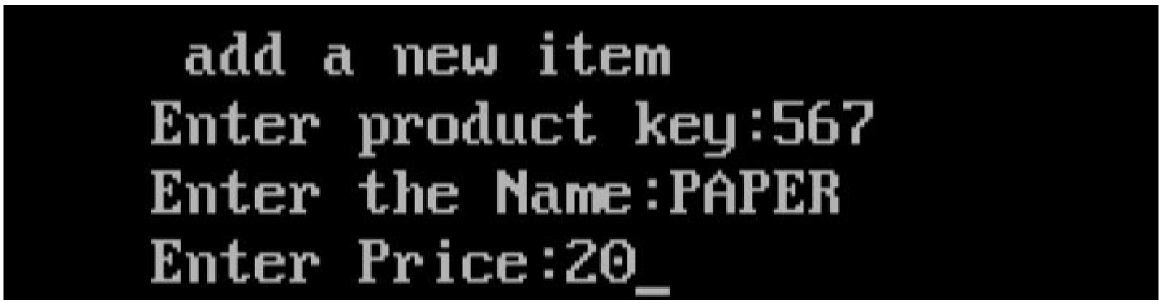


```
 ==============================================
 |         Administrator Menu                 |
 ==============================================
----------------------------------------------
|        1. Add an item                     |
----------------------------------------------
----------------------------------------------
|        2. Search an item                  |
----------------------------------------------
----------------------------------------------
|        3. Display all items               |
----------------------------------------------
----------------------------------------------
|        4. Delete an item                  |
----------------------------------------------
----------------------------------------------
|        5. Modify an item                  |
----------------------------------------------
----------------------------------------------
|        6.Exit                             |
----------------------------------------------
Input number Associated with Action: _
```

Figure 4.2: Admin Men

4.4.3 Adding a record

This is how one adds a record to the file.

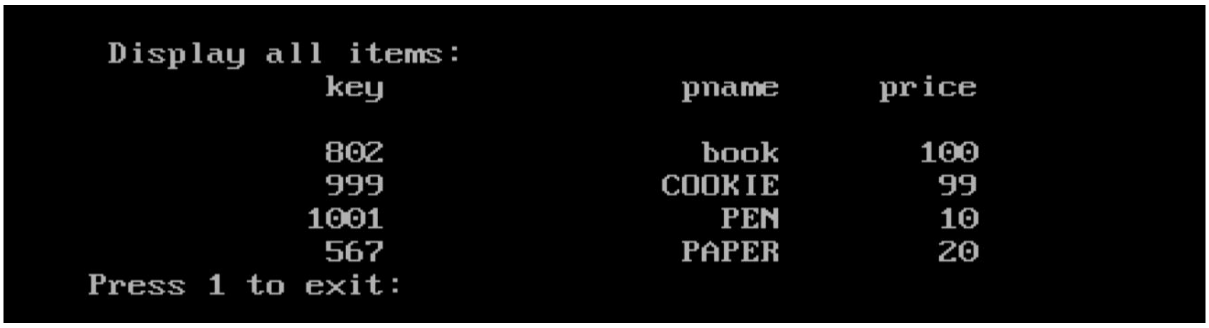


```
  add a new item
Enter product key:567
Enter the Name:PAPER
Enter Price:20_
```

Figure 4.3 Insertion of record

4.4.4 Displaying all the records

All the records stored in the file are displayed to the admin.

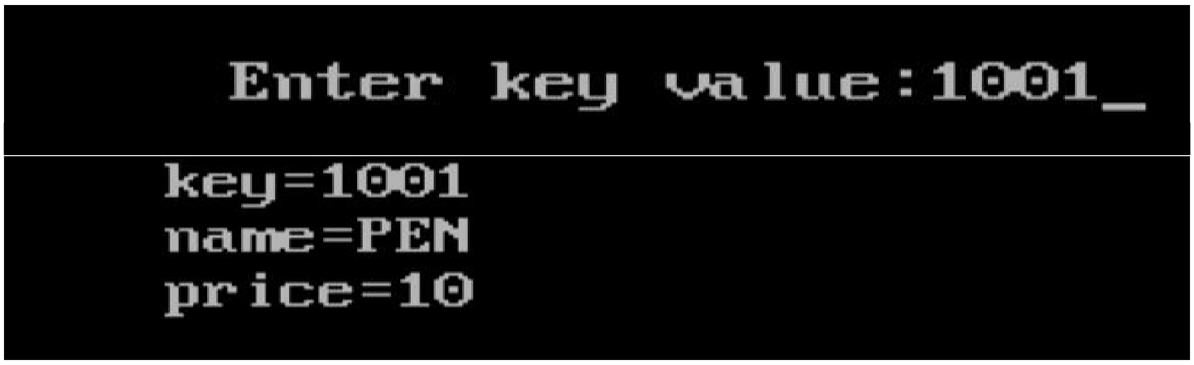


```
 Display all items:
           key                 pname      price

           802                  book        100
           999                COOKIE         99
          1001                   PEN         10
           567                 PAPER         20
Press 1 to exit:
```

Figure 4.4 Displaying all the records

4.4.5 Searching of record

The record with key 1001 is displayed as shown.

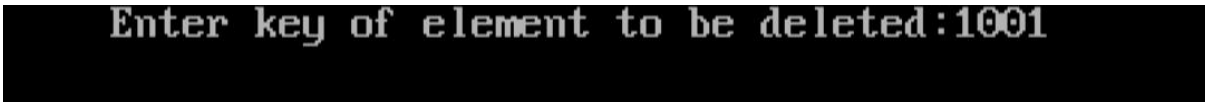


```
Enter key value:1001_

key=1001
name=PEN
price=10
```

Figure. 4.5 Searching for record

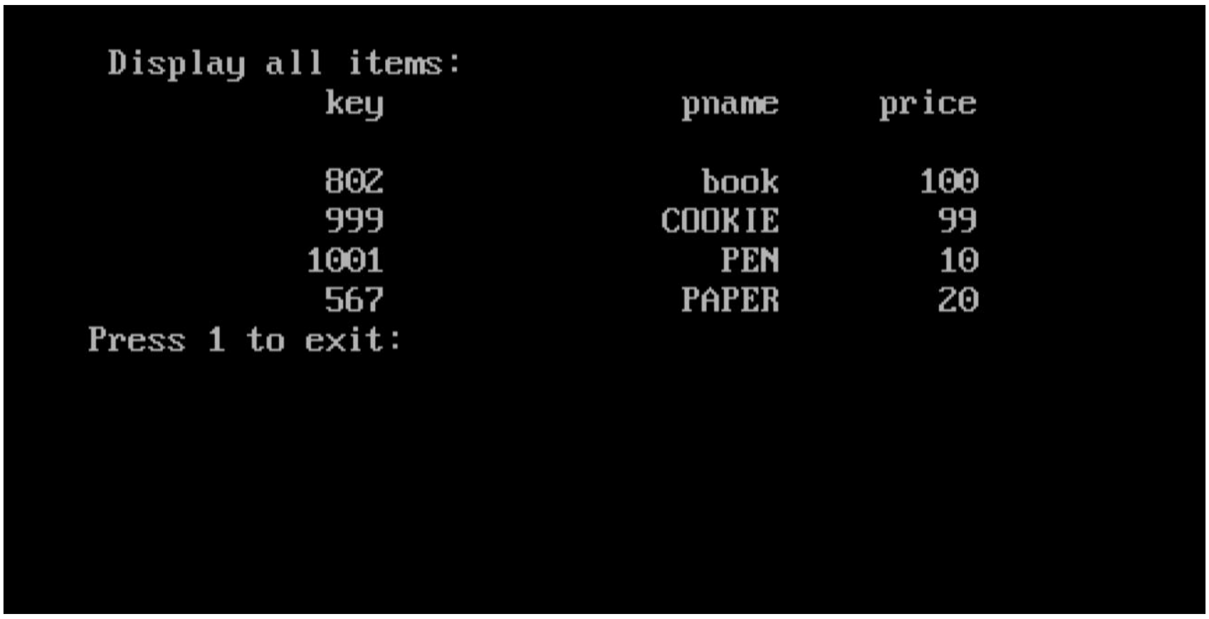
4.4.6 Deletion of record

The process of deletion of a record from the file.



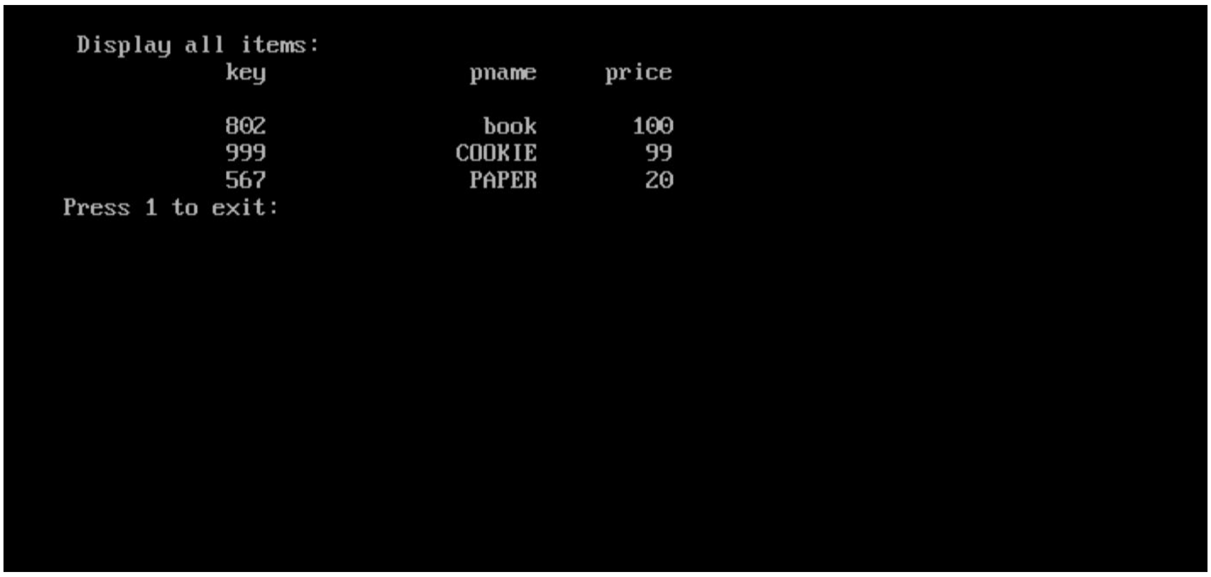
```
Enter key of element to be deleted:1001
```

Figure. 4.6 Deletion of record



```
Display all items:
          key                 pname      price

          802                  book        100
          999                COOKIE         99
         1001                   PEN         10
          567                 PAPER         20
Press 1 to exit:
```

Figure. 4.7 Before deletion of record

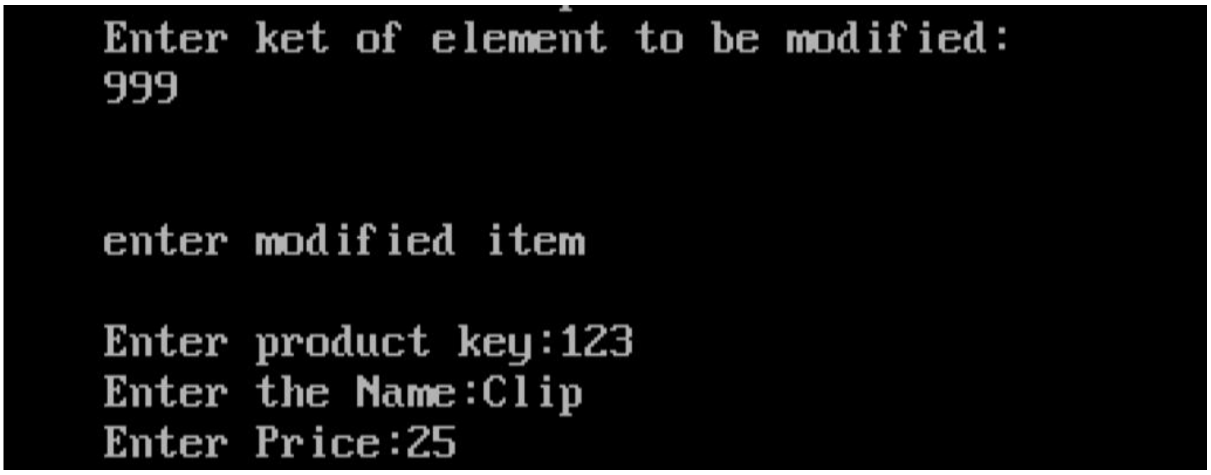


```
Display all items:
          key                pname     price

          802                 book       100
          999               COOKIE        99
          567                PAPER        20
Press 1 to exit:
```

Figure. 4.8 After deletion of record

4.4.7 Modification of record

Record is modified by first deleting the record and followed by addition of new record.

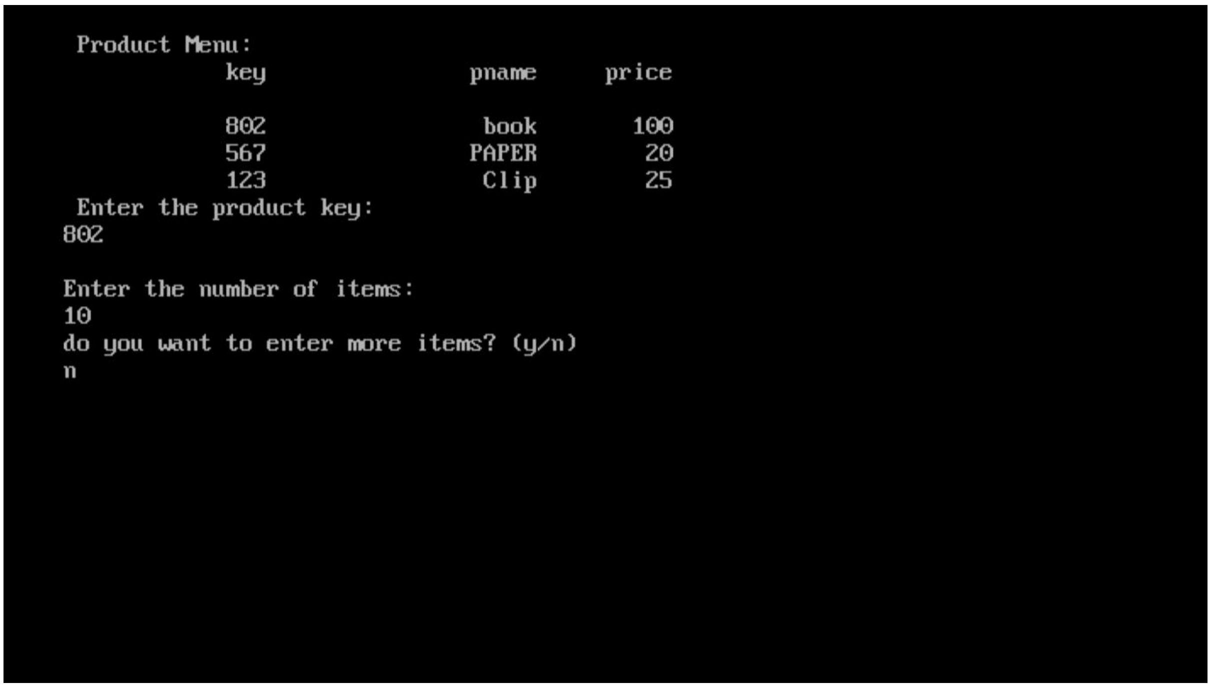


```
Enter ket of element to be modified:
999


enter modified item

Enter product key:123
Enter the Name:Clip
Enter Price:25
```

Figure. 4.9 Modification of record

4.4.8 Staff Menu

This is interface for the staff members who generate the bill.

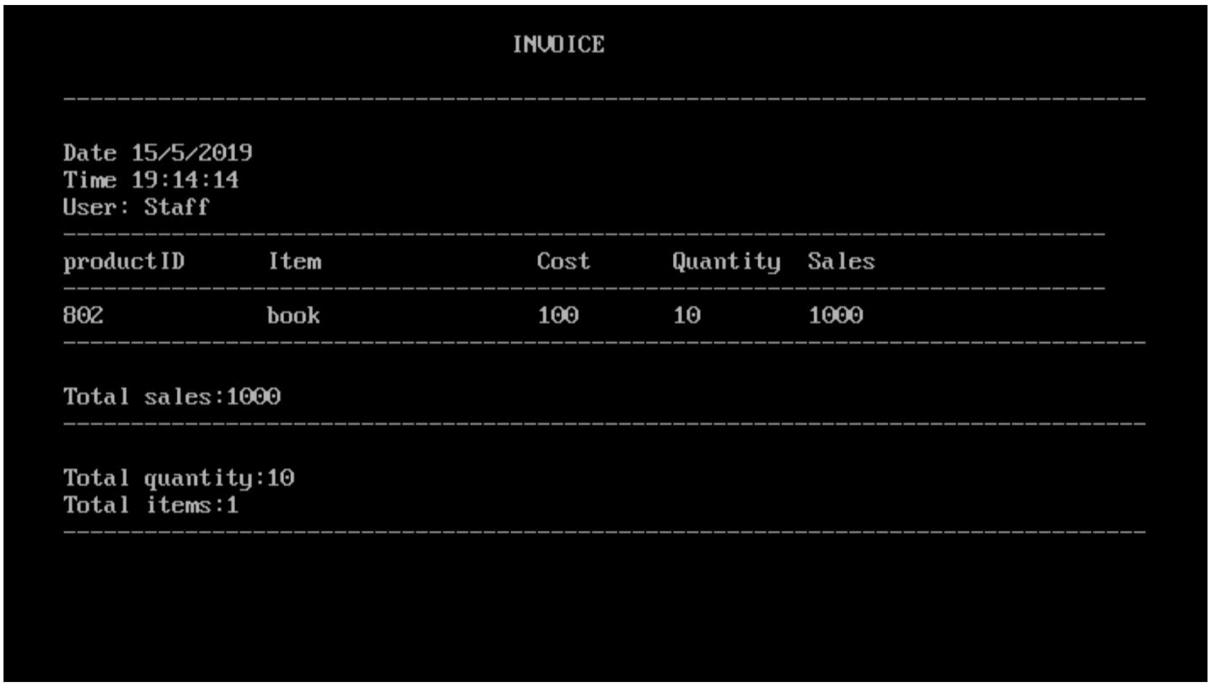


```
 Product Menu:
           key              pname     price

           802               book       100
           567              PAPER        20
           123               Clip        25
 Enter the product key:
802

Enter the number of items:
10
do you want to enter more items? (y/n)
n
```

Figure. 4.10 Staff Menu

4.4.9 Invoice report

This is bill generated for the customer.

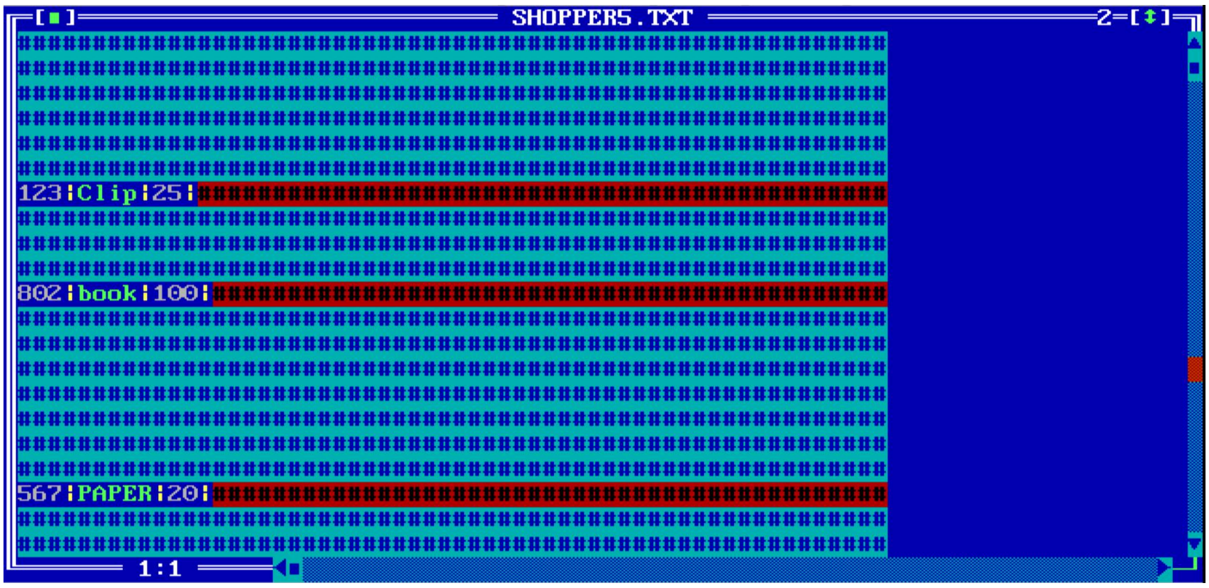


```
                                     INVOICE

--------------------------------------------------------------------------------

Date 15/5/2019
Time 19:14:14
User: Staff
-----------------------------------------------------------------------------
productID      Item               Cost      Quantity  Sales
-----------------------------------------------------------------------------
802            book               100       10        1000
--------------------------------------------------------------------------------

Total sales:1000
--------------------------------------------------------------------------------

Total quantity:10
Total items:1
--------------------------------------------------------------------------------
```

Figure. 4.11 Invoice

4.4.10 File Contents

This is how the data is stored in the hash file.

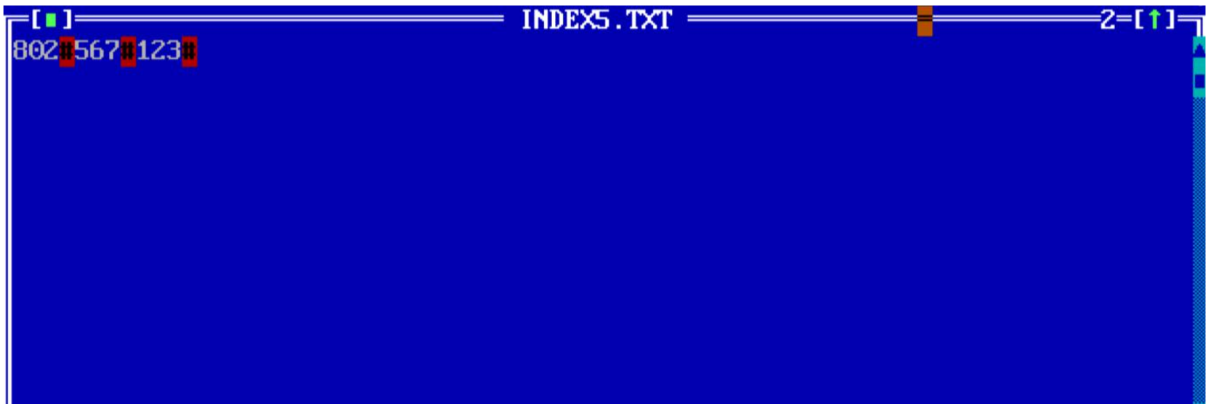


```
##########################################################
##########################################################
##########################################################
##########################################################
##########################################################
##########################################################
123|Clip|25|##############################################
##########################################################
##########################################################
##########################################################
802|book|100|#############################################
##########################################################
##########################################################
##########################################################
##########################################################
##########################################################
##########################################################
##########################################################
567|PAPER|20|#############################################
##########################################################
##########################################################
```

Figure. 4.12 Hash File

4.4.11 Index File Contents

The primary keys are stored in a separate index file as follows.



```
802#567#123#
```

Figure. 4.13 Index File

Chapter 5

CONCLUSION AND FUTURE ENHANCEMENTS

This mini project has successfully been designed and implemented using a Hashing indexing technique to allow to view billing details. This system can be used to add records with its specifications into a file. This system can also be used to search, delete, modify and display existing records stored in a file. The hashing indexing technique has provided faster and direct access to records. The main drawback of hashing is it leads to wastage of memory when they are fewer records.

In future work the system can be implemented using bucket implementation technique to store the data efficiently or other hashing methods can be used such as double hashing ,chained progressive hashing can be used to improve the performance of the system.

REFERENCES

BOOKS

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

[2] K.R.Venugoapal, , K.G.Srinivas, P.M.Krishnaraj:File structures Using C++,TATA McGraw Hill 2008.

[3] www.wikipedia.org

[4] www.w3cSchool.com/php?

[5] www.tutorialspoint.com/sdlc/

[6] www.geeksforgeeks.org

[7] https://forum.directadmin.com/showthread.php?t=15486