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.

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document, it is next inverted to develop an inverted index. Querying the forward index would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document, it is next inverted to develop an inverted index. Querying the forward index

would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

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.

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)
```
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

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.

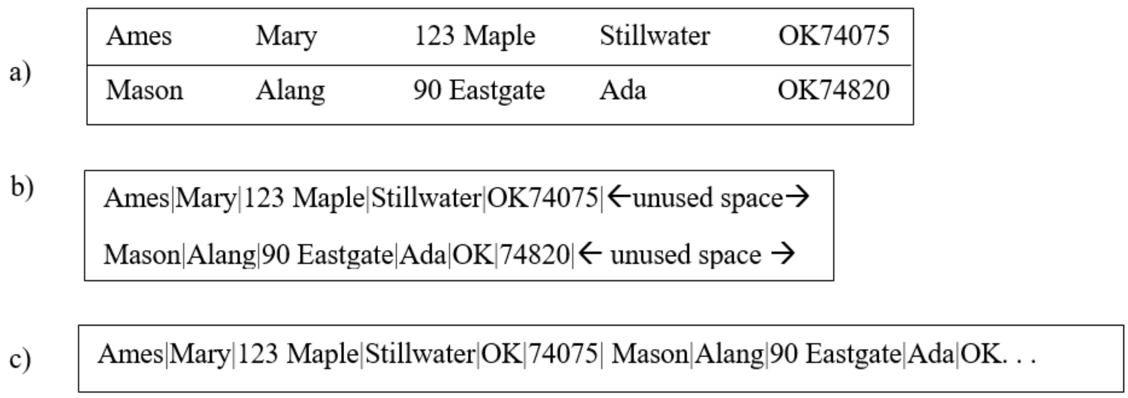
Method 1: Make the Records a Predictable Number of Bytes (Fixed-Length Record)

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. In this Figure below, we have 6 contiguous fields and we can recognize fields simply by counting the fields modulo 6.



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

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.

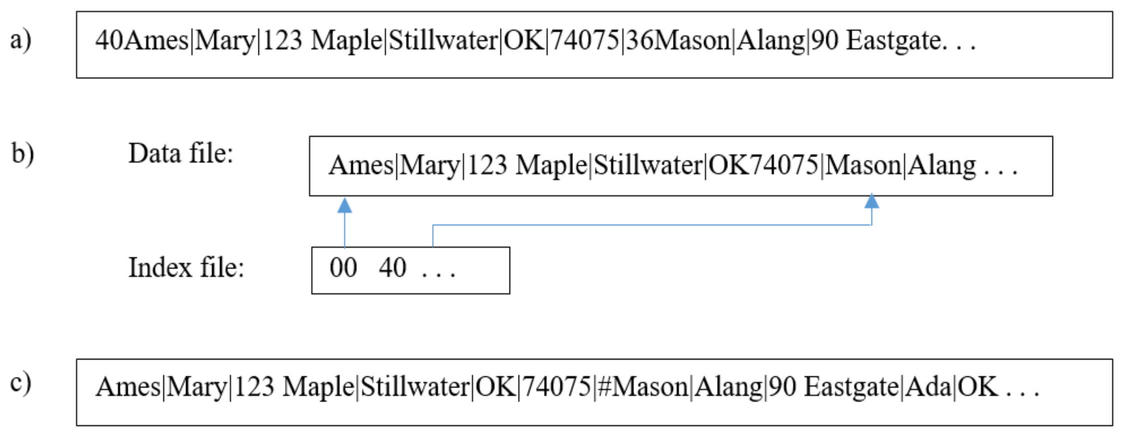

a)
40Ames|Mary|123 Maple|Stillwater|OK|74075|36Mason|Alang|90 Eastgate. . .
b)
Data file:
Ames|Mary|123 Maple|Stillwater|OK74075|Mason|Alang . . .
Index file:
00 40 . . .
c)
Ames|Mary|123 Maple|Stillwater|OK|74075|#Mason|Alang|90 Eastgate|Ada|OK . . .

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. 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) 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, 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 Project

In this project we deal with hotel management system. After storing customer information, room is allocated to the person according to need. All the customer's details are stored in files. Room is allocated by its number. In customer details name, email, phone number is stored securely in files. Multiple rooms can also be allocated depending on the need of customer. The hotel manager can view the details of customer and search based on the names or room number.

2.2 Structure used to Store the Fields and Records

Storing Fields

Fixing the Length of Fields:

In the Hotel Management System, the ROOM_NO 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 -

1. Room_no	Char[5]
2. Name	Char[20]
3. Checkindate	Char[20]
4. email	Char[25]
5. phone	Char[15]

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 Fields:

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.

Using an Index to Keep Track of Addresses:
We use a key-reference pair to keep position for each record in the original file. The positions 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 index file, 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 '#' and followed by end-of-line (new-line) character (\n).

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 room number and details of customer and other fields entered, in the data file. Records with duplicate room number fields are not allowed to be inserted.

- **Display**

The system can then be used to display existing records of all customers. The records are displayed based on the ordering of position in the data file. Only records with references in the inverted list are displayed. This prevents records marked as deleted (using '$') from being displayed.

- **Search**

The system can then be used to search for existing records of a customer. The user is asked to input the name they wish to search and then the list of room numbers allotted to the name is shown using inverted list and user is asked to input the room number they want to search. The room details are then shown based on the room number input by 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 '$' symbol is placed at the starting of the record that the user selected to be deleted. The user gives the room no of the record that needs to be deleted. If absent, a "record not found" message is displayed to the user.

- **Modify**

The system can then be used to modify 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 at the start of record, to help distinguish it from records that should be displayed. 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 room number.

2.4 Indexing Used

Inverted List

In computer science, an inverted index (also referred to as a postings file or inverted file) is a database index storing a mapping from content, such as words or numbers, to its locations in a table, or in a document or a set of documents (named in contrast to a forward index, which maps from documents to content). The purpose of an inverted index is to allow fast full-text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines. Additionally, several significant general-purpose mainframe-based database management systems have used inverted list architectures.

The room number and their positions are stored in the index file. Inverted indexing is done based on the names in which names of the customers are stored in the file and with it the corresponding room number is stored against each name. When we modify, insert or delete a record the index file is updated for the same and with that the inverted index file is also updated. Through inverted index file we are able to search using the name of given customer and know the room details of the customer using the corresponding room number.

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 five fields. There are:

1. Room_no with length 5
2. Name with length 20
3. phone with length 15
4. email with length 25
5. Checkindate with length 20

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 like

- room_no which is of the type char and length from 0 to 5
- Name which is of the type char and length from 0 to 20
- Phone number which is of the type char and length from 0 to 15
- E-mail which is of the type char and length from 0 to 25
- Check in date which is of the type char and length from 0 to 20

3.2.2 Deletion of Record

In the deletion operation the user need to enter 4. It will take the user to a new page in which it will show the user all the room no. Once the room no is entered by the user which he want to delete then 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 data file at the start of the deleted record there will be ‘$’ placed indicating that the record has been deleted.

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 room_no, name, phone, Email ID, Check in date.

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 enter the program will take the user to a new page in which the Name to be searched need to be entered. Once entering it, the user will be shown room numbers according to the name and then ask the user to enter the room number, 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.

3.2.5 Modifying of Records

In modify operation, if the user need to modify the existing details then the user need to enter 3 according to the menu bar. In this process the user need to enter the room number to which modification need to be done. In case the room_no is found, then the user can make modification to the already existing details like the name, phone, email ID, check in date and even the room_no.

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 *hotel* object with the Room number, email id, phone number & check in date, 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. 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 class hotel dynamically, using the method *malloc(),*which returns a pointer (or reference), to the allocated block. Pointers are also data members of objects 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.

4.1.4 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 (Figure 4.1) adds index objects to the Hashing if the ROOM_NO 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.

```
void hotel::read()
{
        cout<<"\n\n\n";
        cout<<"Enter the Room Number : ";
        gets(room_no);
        while(searchDup(room_no))
        {
                cout<<"\n\nDuplicate Entry please enter room number again : ";
                gets(room_no);
        }
        strcpy(allroom[totroom],room_no);
        //allroom[totroom]=room_no;
        totroom++;
        cout<<"\n\nEnter the Name : ";
        gets(name);
        cout<<"\n\nEnter the Phone Number : ";
        gets(phone);
        cout<<"\n\nEnter the Email address : ";
        gets(email);
        cout<<"\n\nEnter the Check in Date(dd/mm/yy) : ";
        gets(checkindate);
        pack();
        clrscr();
}
```

4.2.2 Display Module Pseudocode

The display function in the objects are used to retrieve and display the corresponding data record fields.

```
void hotel::display()
{
        while(1)
        {
                unpack();
                if(ifile.eof())
                        break;
                if(room_no[0] != '$')
                {
                cout<<"\n\n\n\n\n";
                cout<<"\tRoom Number      : "<<room_no;
                cout<<"\n\n\tName         : "<<name;
                cout<<"\n\n\tPhone        : "<<phone;
                cout<<"\n\n\tEmail        : "<<email;
                cout<<"\n\n\tCheck in Date : "<<checkindate;
                getch();
                clrscr();
                }
        }
}
```

4.2.3 Deletion Module Pseudocode

The *deletion()* function (Figure 4.3) deletes values from nodes and balances the inverted list files after. It makes calls to other recursive and non-recursive functions.

```
void hotel::deleteRec(int flag)
{
        ifile.seekg(flag,ios::beg);
        ifile.put('$');
        cout<<"\n\nRecord deleted!";
}
```

4.2.4 Search Module Pseudocode

The *search()* function (Figure 4.4), finds the record for the corresponding room_no and then returns a corresponding value.

```
int hotel::search()
{
        char sroom_no[5],dummy[100];
        int flag;
        fstream nfile(filename,ios::in);
        cout<<"\n\nList of rooms currently alloted : \n\n";
        while(!nfile.eof())
        {
                char rooms[5];
                nfile.getline(rooms,5,'|');
                nfile.getline(dummy,100,'#');
                if(rooms[0] != '$')
                cout<<rooms<<"\t";
        }
        nfile.close();
        char ch='y';
        cout<<"\n\nDo you still want to update or delete?(y or n)\n\n";
        cin>>ch;
        if(ch == 'n' || ch == 'N')
                return -1;
        cout<<"\n\nEnter the room number : ";
        gets(sroom_no);
        while(!ifile.eof())
        {
                flag=ifile.tellg();
                unpack();
                if(strcmpi(room_no,sroom_no)==0)
                {
                        cout<<"\n\nRecord found!!";
                        return flag;
                }
        }
        return -1;
}
```

4.2.5 Update module pseudocode

There can be few situations where a record that is already present should be updated. The fields have to be updated as per the users requirements. The user is asked for a key and is matched from the existing records. If the record is found the user can update the fields he wants to.

```
void hotel::update(int flag)

{

        ifile.seekp(flag,ios::beg);

        ifile.put('$');

        ifile.seekp(0,ios::end);

        read();

        }
```

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

1. Room no. input is checked to see if it is of integers only:

- digits for all characters.

Table 4.1 Unit Test Cases for Hotel record details

Case id	Description	Input Data	Expected Output	Actual Output	Status
1.	Enter valid RoomNo	12	Press Return for next value	Press Return for next value	Pass
2	Enter invalid RoomNo	I12	Wrong RoomNo format.	Wrong RoomNo format	Pass
3	Enter Name	Ashish	Press return for next value	Press return for next value	Pass
4	Enter Phone	9900712684	Press return for next value	Press return for next value	Pass
5	Enter Email	ashish@ gmail.com	Press return for next value	Press return for next value	Pass
6	Enter Check in date	23/4/19	Press return for next value	Press return for next value	Pass

4.3.2 Functional Testing

Functional testing is a software testing process used within software development in which software is tested to ensure that it conforms with all requirements. Functional testing is a way of checking software to ensure that it has all the required functionality that's specified within its functional requirements.

Functions (or features) are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied by the application. This type of testing is not concerned with how processing occurs, but rather, with the results of processing. During functional testing, Black Box Testing technique is used in which the internal logic of the system being tested is not known to the tester.

Table 4.2 Functional Test Cases for Room No

Case id	Description	Input Data	Expected Output	Actual Output	Status
1	Opening a file to insert the data	Insert option is selected	File should be opened in append mode without any error	File opened in append mode	Pass
2	Validating RoomNo	12	Accept the RoomNo and prompt for name will be displayed	“Name : ”	Pass
3	Validating RoomNo	I12	Invalid RoomNo and prompt to enter RoomNo again	“Room No. : ”	Pass
4	Room No : Name : Phone : Email : Check in date :	12 Ashish 9900612784 ashish@gmail.co 23/4/19	It should accept all the inputs and should return to main menu	“Press any key to return to main menu”	Pass
5	File updating	-	It should pack all the fields and will be appended to the data file	Data will be updated both in data file and index file	Pass
6	Closing a file	-	File should be closed without any error.	File is closed	Pass
7	Check for updated in file	All records inserted updated or not	File should be updated and records are appended at end of the file	Same as expected output	Pass

4.3.3 Integration Testing

Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration testing is also taken as integration and testing this is the major testing process where the units are combined and tested. It's main objective is to verify whether the major parts of the program is working fine or not. This testing can be done by choosing the options in the program and by giving suitable inputs it is tested.

Table 4.3 Integration testing for all modules

Case id	Description	Input Data	Expected Output	Actual Output	Status
1	To display entered records of the data file	Enter the option 2 in the hotel menu	Displays the records one after the other	Displays the records one after the other	Pass
2	To add the new records into the data file	Enter the option 1 in the hotel menu	Display the record entry form	Display the record entry form	Pass
3	To search for a particular record in the file	Enter the option 2 in the main menu and should enter the I12	Record not found.	Record not found.	Pass
4	To search for a particular record in the file	Enter the option 2 in the main menu and should enter the 12	'Record is found' and it will display the contents of the searched record.	Same as expected output.	Pass
5	To delete particular record in the file	Enter the option 4 in the hotel menu and should enter the I12	Record not found.	Record not found.	Pass

6	To delete particular record in the file	Enter the option 4 in the hotel menu and should enter the 12	Delete the record and both data file and index file will updated.	Delete the record and both data file and index file will updated.	Pass
7	To update particular record in the file	Enter the option 3 in the hotel menu and should enter the Room NO I12	Record not found	Record not found	Pass
8	To update particular record in the file	Enter the option 3 in the hotel menu and should enter the Room No 12	'Record is found' and delete the record and allow the users to re-enter the record.	'Record is found 'and delete the record and allow the users to re-enter the record.	Pass
9	To display the inverted index structure of the entered records	Enter the option 1 in the indexing menu	Displays the inverted index structure of the records	Displays the inverted index structure of the records	Pass
10	To quit the program	Enter the option 3 in the main menu	Exit the program	Exit the program	Pass

4.3.4 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.

Table 4.4 System Testing for Hotel record details

Case id	Description	Input data	Expected output	Actual output	Status
1	To display the records	Display records for valid roomno=‘12’(present)and invalid roomno =’123’(not present)	Record is displayed for valid and record not found for invalid roomno	Record is displayed for valid roomno and record not found for invalid roomno	Pass
2	To search the records	Display records for valid roomno=‘12’(present)and invalid roomno =’123’(not present)	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 roomno = ‘12’(present) and invalid roomno=’123’(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 roomno. = ‘12’(present) and invalid accno = ’123’(not present)	Record is Updated for valid and record not found for invalid	Record is Updated for valid and record not found for invalid	Pass

4.3.5 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. [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.

Table 4.5 Acceptance Testing for Hotel record details

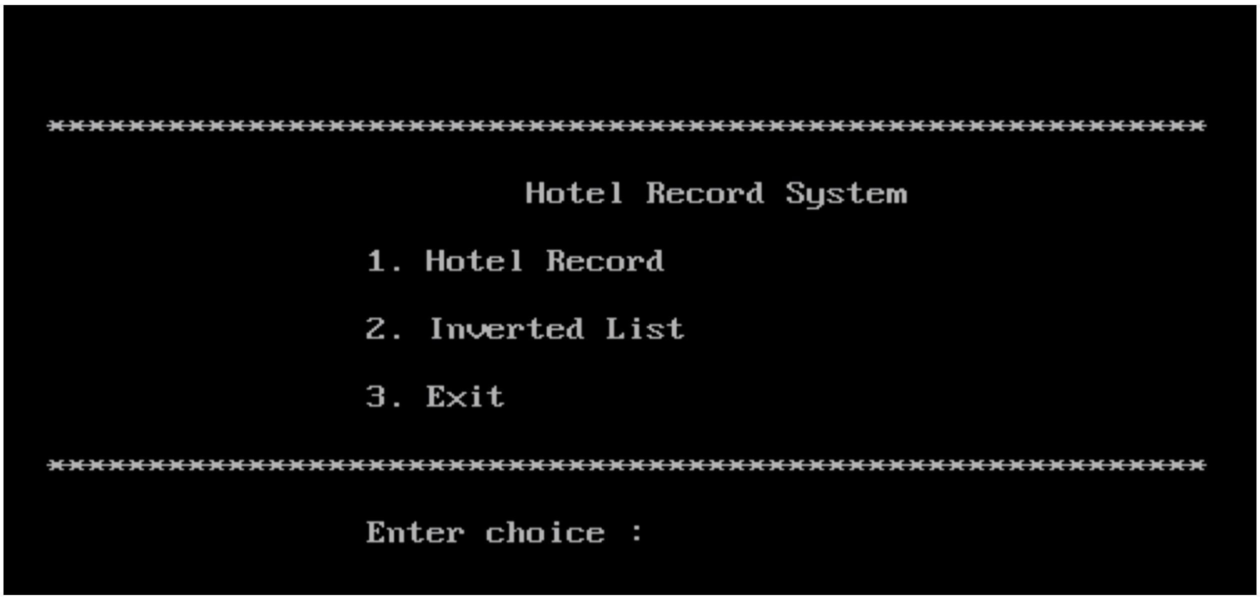
Case ID	**Description**	**Input Data**	**Expected Output**	**Actual Output**	**Status**
1.	Insertion Module	Inserting valid data	Record added successfully	Record added successfully	Pass
2.	Deletion Module	Enter valid roomno.	Record deleted successfully	Record deleted successfully	Pass
3.	Updatation Module	Enter valid roomno.	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 4.13

4.4.1 Menu Options

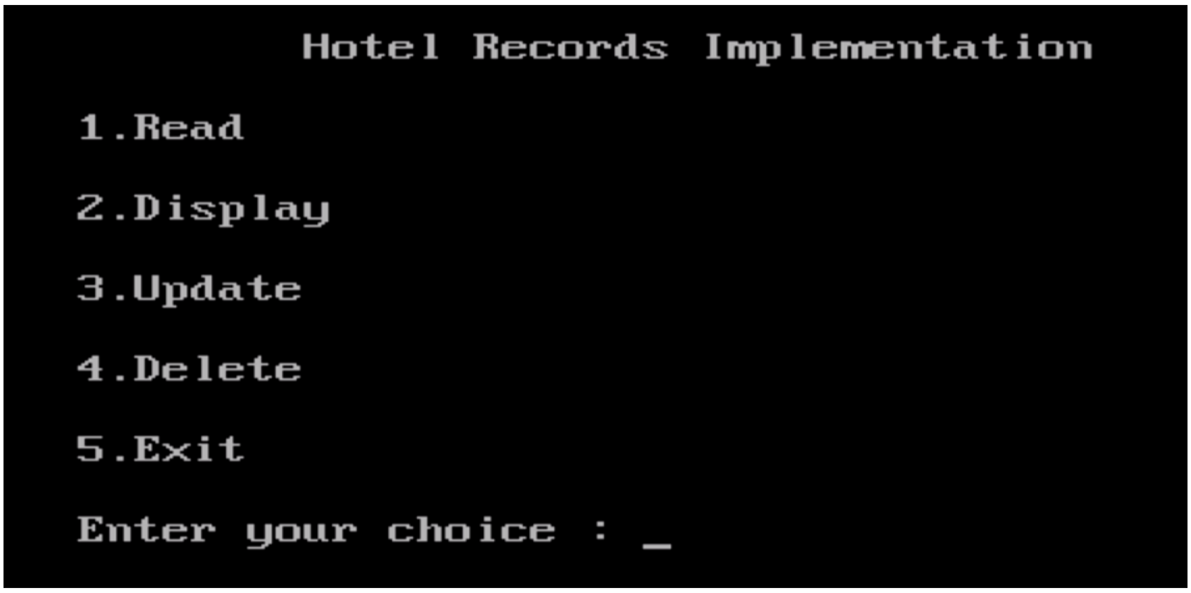
Menu options display the operation to perform like display all records, add record, search, delete, update and quit as shown in figure 4.1, 4.2 and 4.3.



```
************************************************************

                    Hotel Record System

           1. Hotel Record

           2. Inverted List

           3. Exit

************************************************************

           Enter choice :
```

Figure 4.1 User Menu Screen

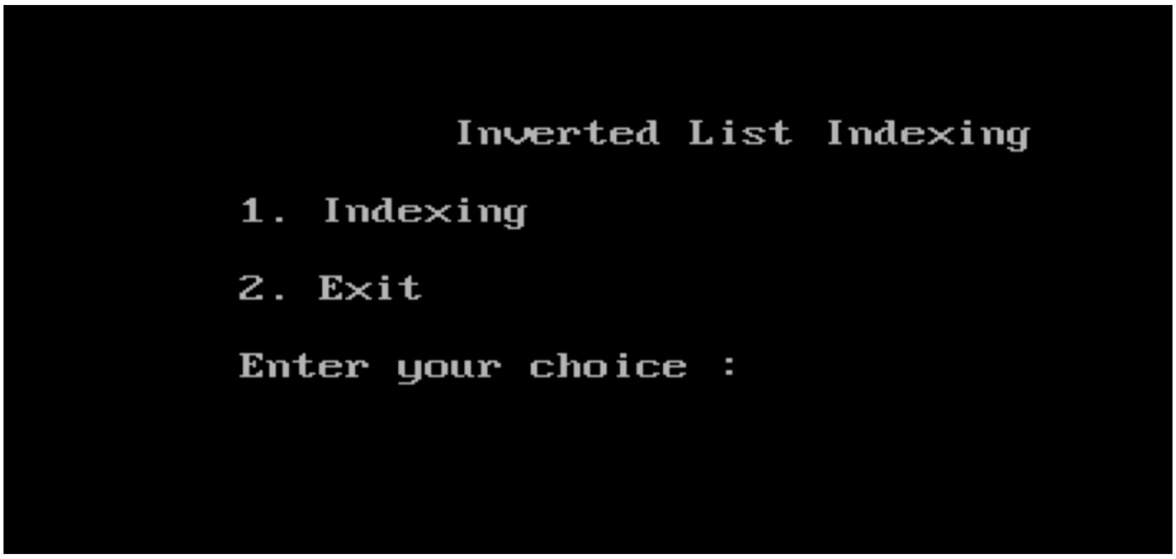
Three options to select whether you want to search or perform other operations.



```
          Hotel Records Implementation

   1.Read

   2.Display

   3.Update

   4.Delete

   5.Exit

   Enter your choice : _
```

Figure 4.2 Hotel Records Implementation

The user can perform operations like read, update, delete and display.



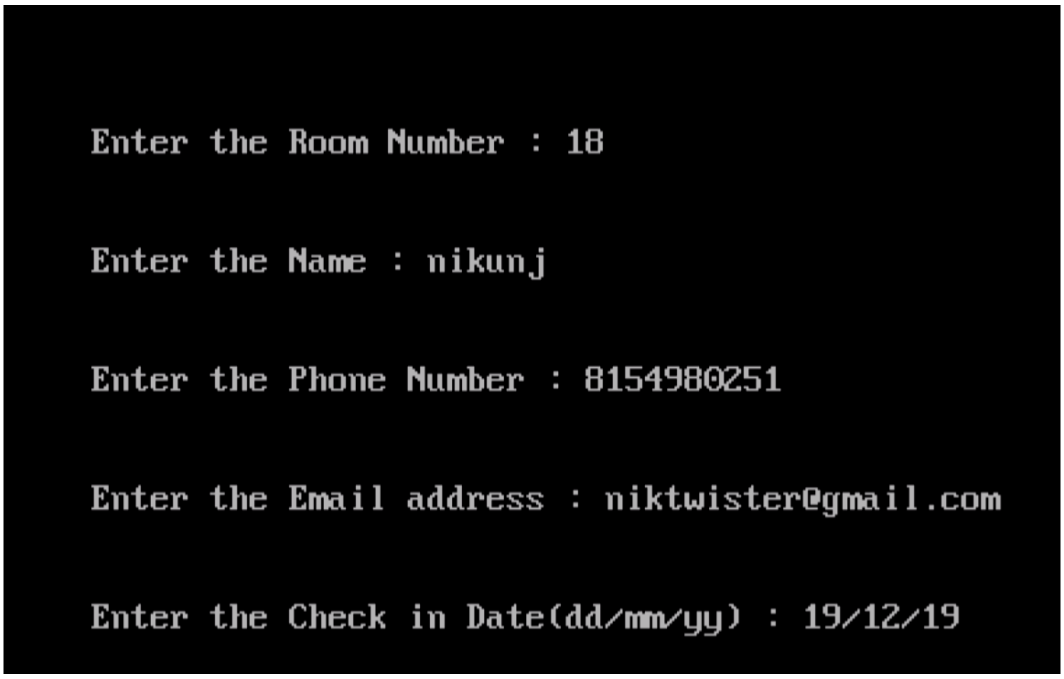
```
                Inverted List Indexing

        1. Indexing

        2. Exit

        Enter your choice :
```

Figure 4.3 Indexing of records based on secondary index

Here record is searched based on name and then displayed on screen.

4.4.2 Insertion

Here record is inserted and room number name phone number email and check in date are updated.

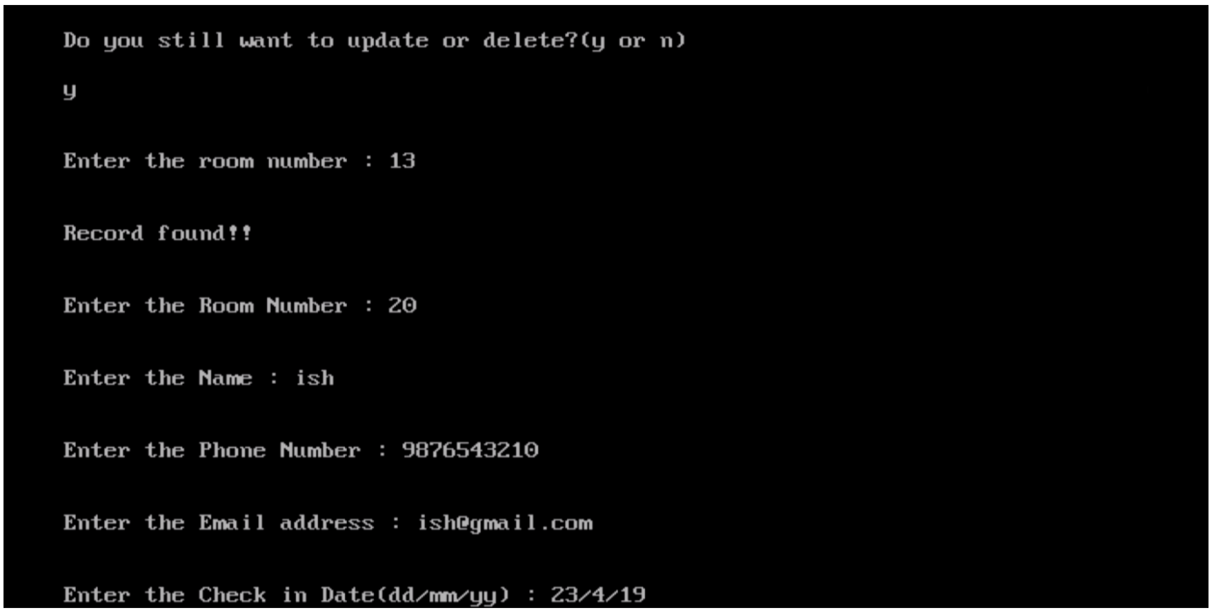


```
Enter the Room Number : 18

Enter the Name : nikunj

Enter the Phone Number : 8154980251

Enter the Email address : niktwister@gmail.com

Enter the Check in Date(dd/mm/yy) : 19/12/19
```

Figure 4.4 Inserting Customer Record.

4.4.3 Updation

Here the record to be modified is searched by using the room number and then user inputs the details that needs to be changed in the record.

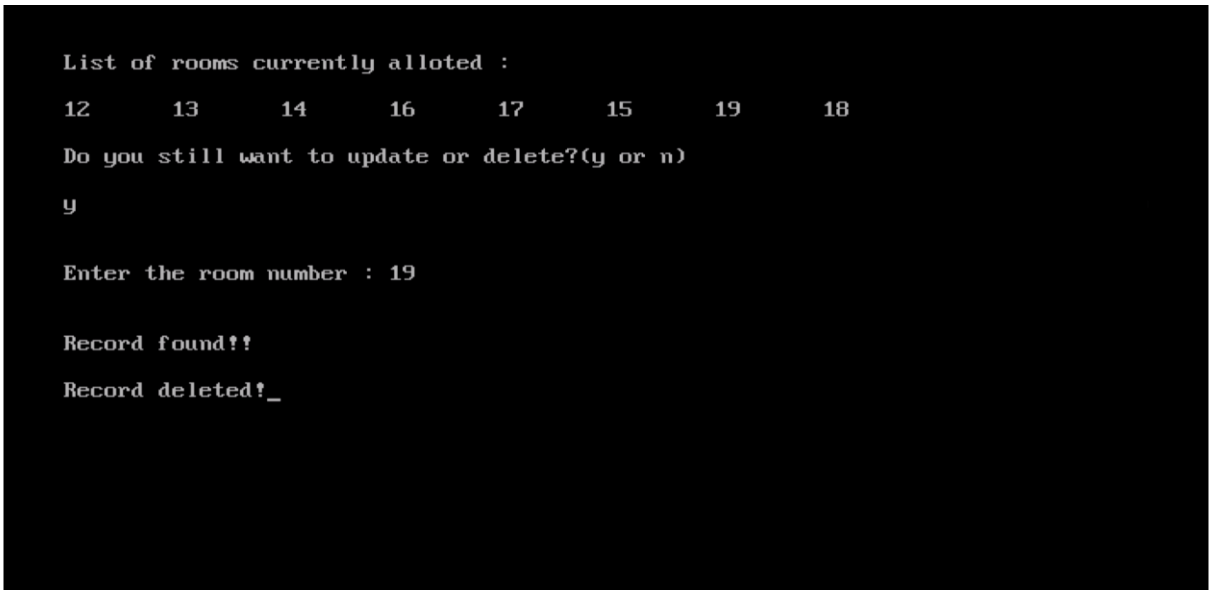


```
Do you still want to update or delete?(y or n)

y

Enter the room number : 13

Record found!!

Enter the Room Number : 20

Enter the Name : ish

Enter the Phone Number : 9876543210

Enter the Email address : ish@gmail.com

Enter the Check in Date(dd/mm/yy) : 23/4/19
```

Figure 4.5 Modifying Customer Record.

4.4.4 Deletion

Here the deletion of record is done by showing the list of room numbers and then user inputs the room number that is to be deleted.

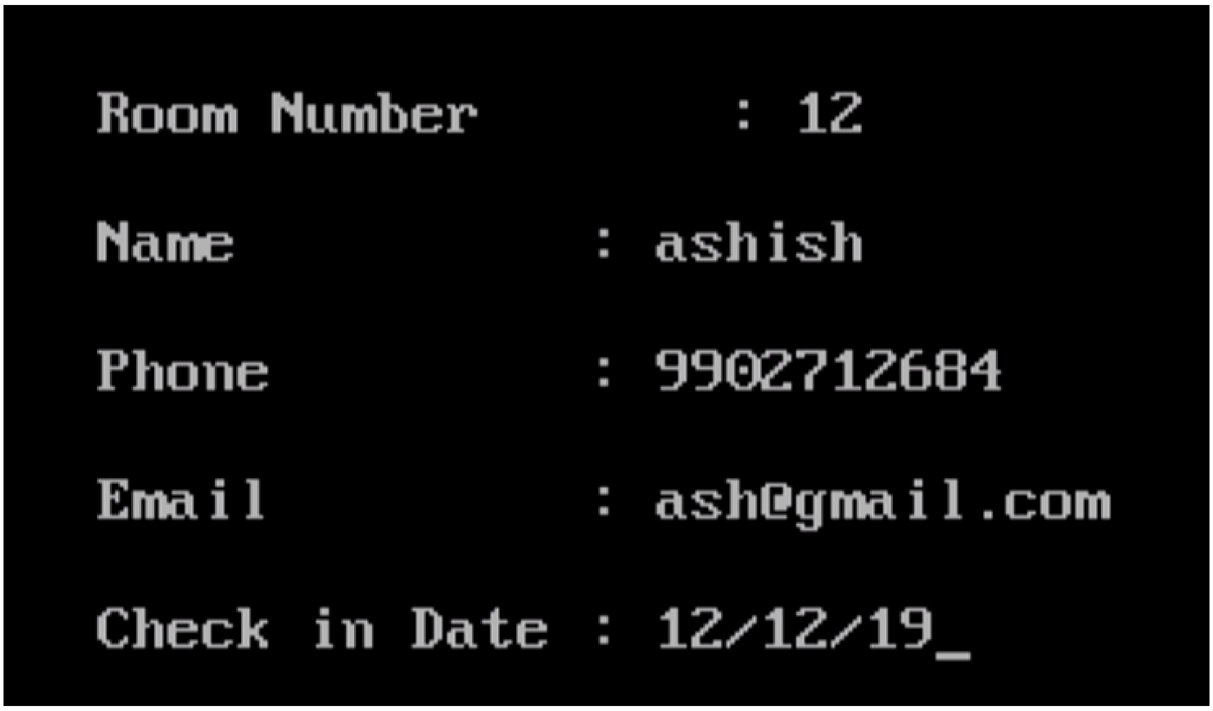


```
List of rooms currently alloted :
12      13      14      16      17      15      19      18
Do you still want to update or delete?(y or n)
y

Enter the room number : 19

Record found!!
Record deleted!_
```

Figure 4.6 Deleting Customer Record

4.4.5 Display

Here the records are displayed one by one, users press enter to access the next record that is displayed on the screen and after that redirected back to the main screen.

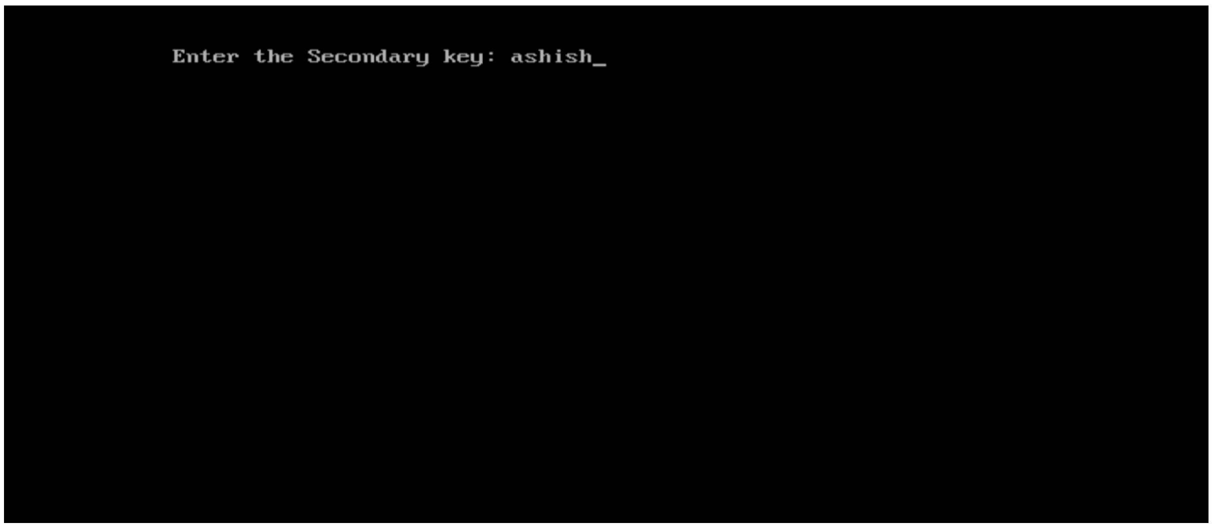


```
Room Number       : 12
Name          : ashish
Phone         : 9902712684
Email         : ash@gmail.com
Check in Date : 12/12/19_
```

Figure 4.7 Snapshot of display of records

4.4.6 Searching

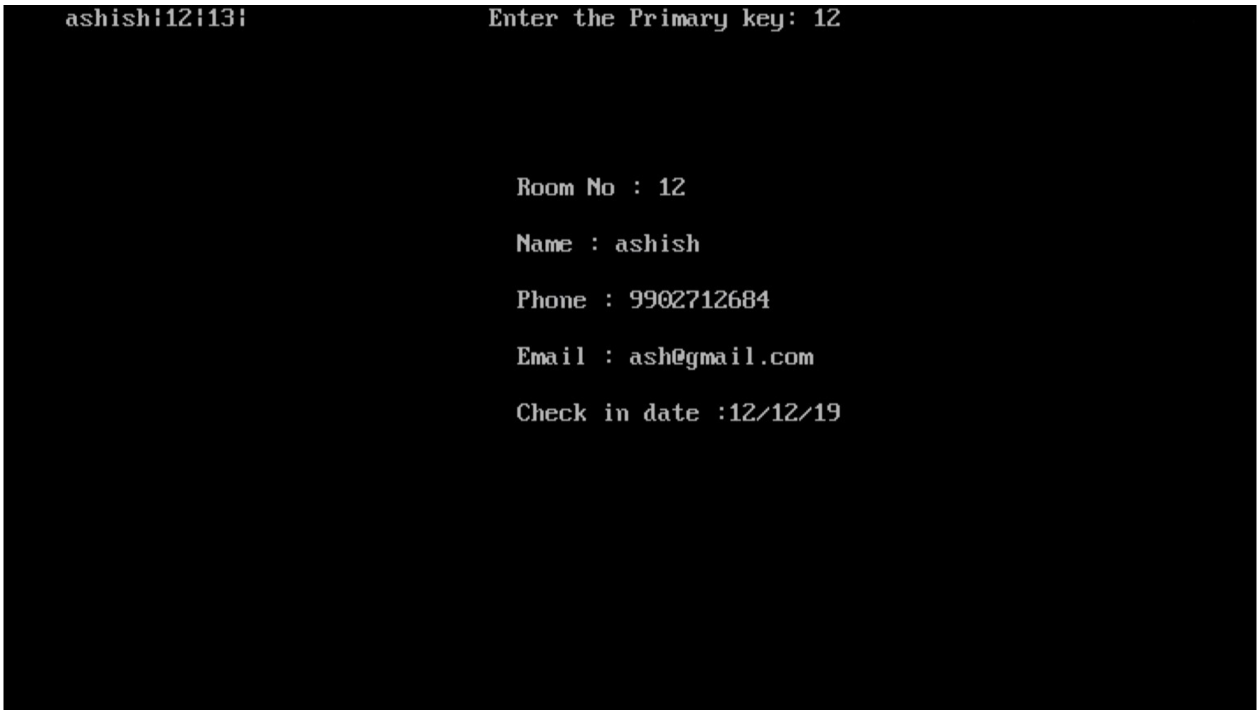
Searching the room number based on secondary index which is name of the customer and then user is shown the room number to select from as shown in figure 4.8 and 4.9.



```
Enter the Secondary key: ashish_
```

Figure 4.8 Snapshot of displaying of the inverted lists

Displaying the record based on the room number selected by the user which is the primary index in the record.

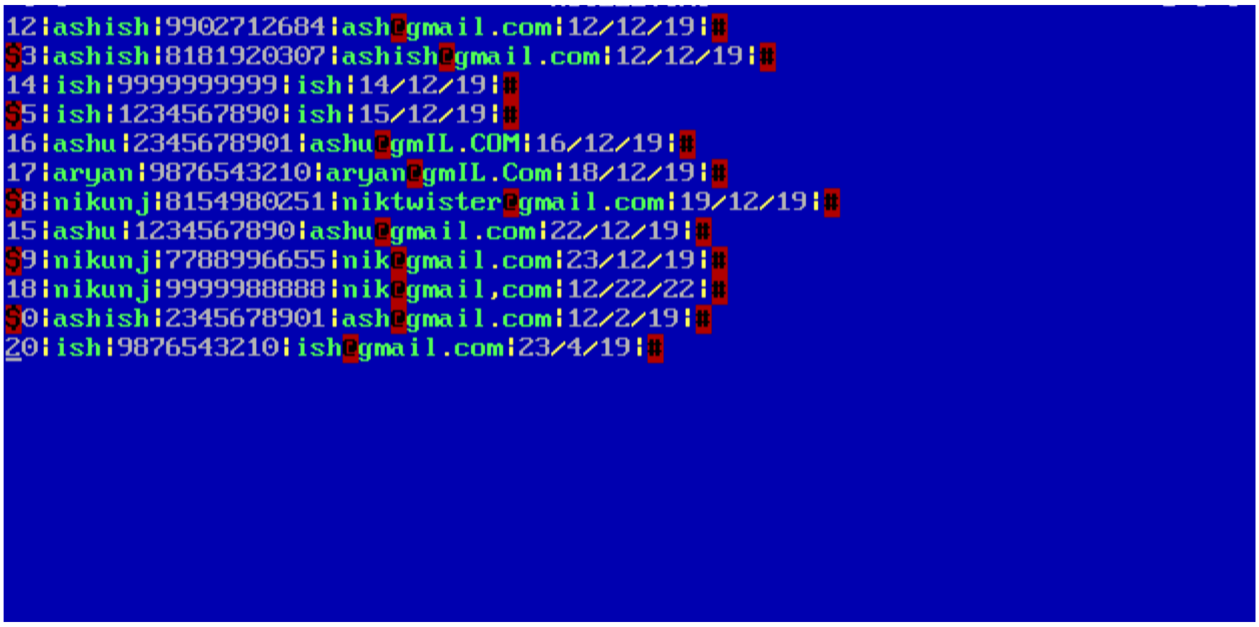


```
ashish¦12¦13¦                Enter the Primary key: 12



                               Room No : 12

                               Name : ashish

                               Phone : 9902712684

                               Email : ash@gmail.com

                               Check in date :12/12/19
```

Figure 4.9 Snapshot of displaying of the inverted lists with primary keys having same Name

4.4.7 File Contents

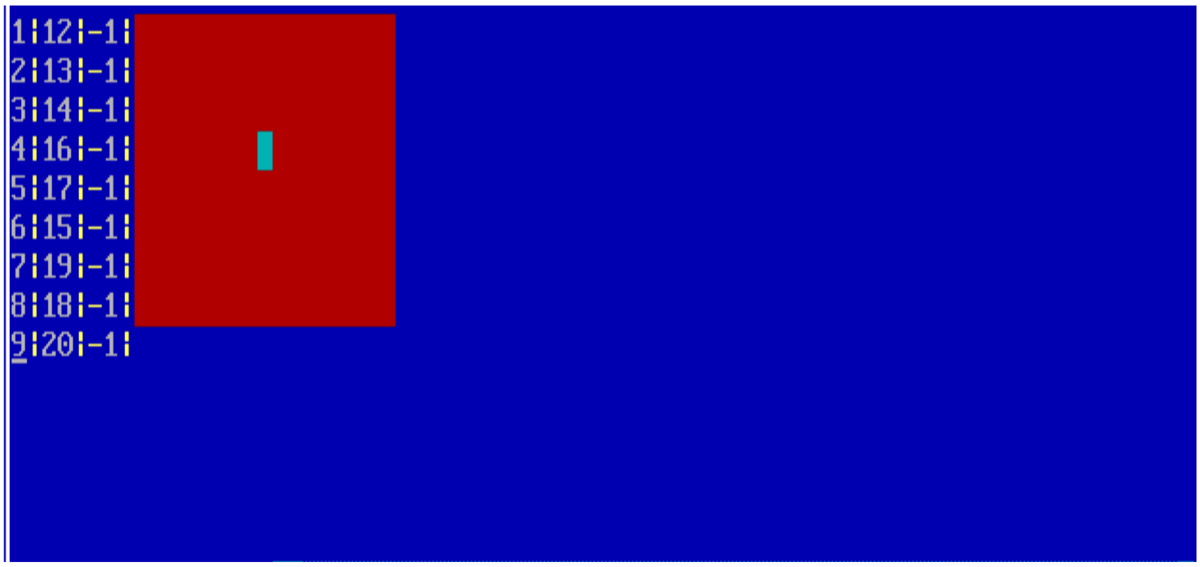
Here the contents of file containing all the records deleted as well as updated and the contents of primary and secondary files is shown in the figure 4.10, 4.11, 4.12 and 4.13. It contains all the records with delimiter '|' and '$' infront of every deleted record.



```
12¦ashish¦9902712684¦ash@gmail.com¦12/12/19¦#
$3¦ashish¦8181920307¦ashish@gmail.com¦12/12/19¦#
14¦ish¦9999999999¦ish¦14/12/19¦#
$5¦ish¦1234567890¦ish¦15/12/19¦#
16¦ashu¦2345678901¦ashu@gmIL.COM¦16/12/19¦#
17¦aryan¦9876543210¦aryan@gmIL.Com¦18/12/19¦#
$8¦nikunj¦8154980251¦niktwister@gmail.com¦19/12/19¦#
15¦ashu¦1234567890¦ashu@gmail.com¦22/12/19¦#
$9¦nikunj¦7788996655¦nik@gmail.com¦23/12/19¦#
18¦nikunj¦9999988888¦nik@gmail,com¦12/22/22¦#
$0¦ashish¦2345678901¦ash@gmail.com¦12/2/19¦#
20¦ish¦9876543210¦ish@gmail.com¦23/4/19¦#
```

Figure 4.10 File contents containing record

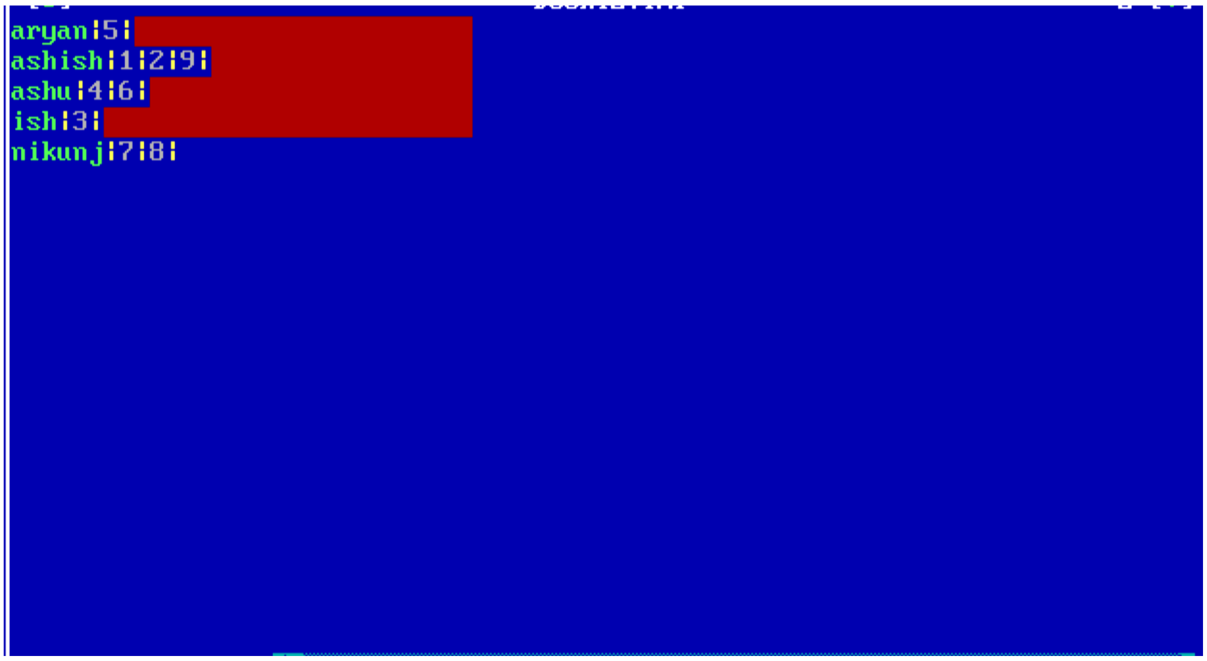
This file contains the primary key and their position where the record is present in the main data file.



```
1|12|-1|
2|13|-1|
3|14|-1|
4|16|-1|
5|17|-1|
6|15|-1|
7|19|-1|
8|18|-1|
9|20|-1|
```

Figure 4.11 Primary file containing the room number and their position.

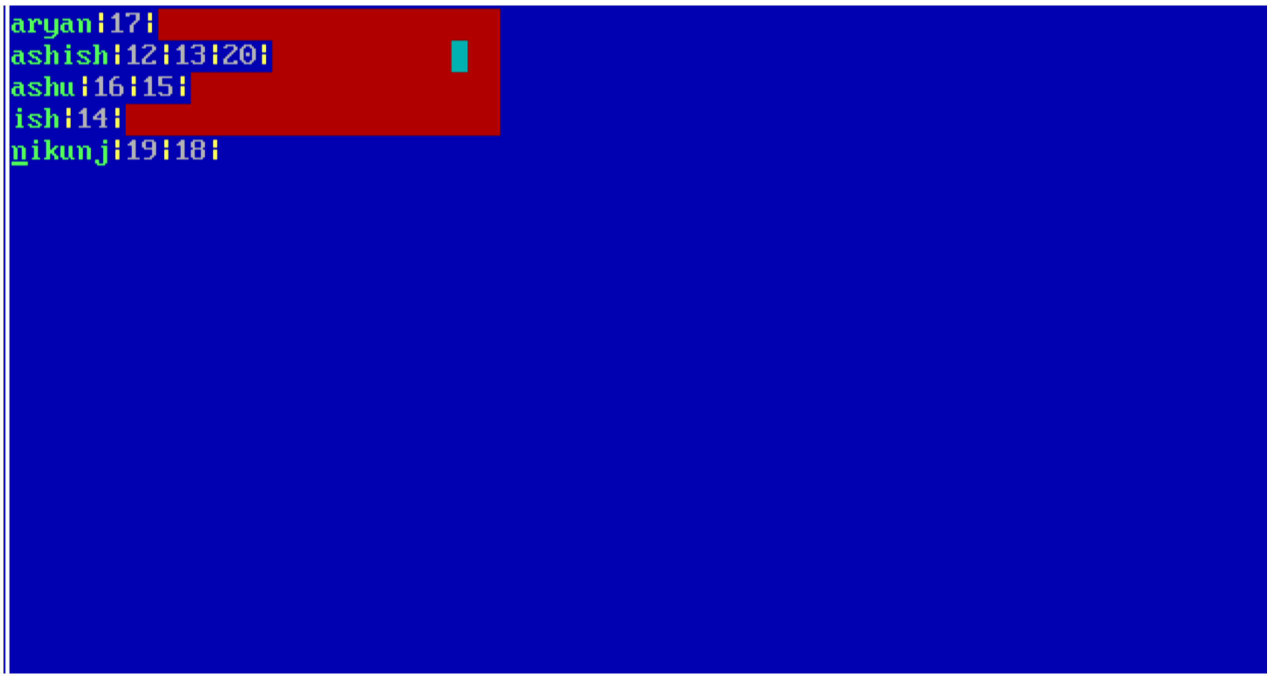
This file contains the secondary keys matched to their position in the record file linked with the name of the customer.



```
aryan|5|
ashish|1|2|9|
ashu|4|6|
ish|3|
nikunj|7|8|
```

Figure 4.12 Secondary file 1 containing the name and their position.

This file contains the secondary keys matched to their respective room numbers in the form of inverted list.



```
aryan|17|
ashish|12|13|20|
ashu|16|15|
ish|14|
nikunj|19|18|
```

Figure 4.13 Secondary file 2 containing the name and their room numbers.

Chapter 5

Conclusion and Future Enhancements

This project achieved the goals that we had set. We have gained the complete details of the Hotels information and many of the projects are working for the welfare-ment of the System. And many apps are formed up to get the details of the Hotels. We have studied in depth the inverted file indexing algorithm and related data structures like hash table. From the performance analysis of the inverted index file indexing algorithm and data structures we can conclude that efficient algorithms and data structures are the key to efficient bus information system. This work also enumerates future work based on this project. Different record structures and field structures can be used to increase efficiency and performance

This project has only added Room Id, further we can also add the details of the Rooms along with selection of its type. If there is a complaint regarding the Rooms or services can be informed to the Hotel management using the details of the Rooms ID we have provided. Since this system will play a key role in the organization, thus success over long period of and reliance to Hotel management system. Further if this project will be uploaded, it can be used to Hotel management system. It can increase the caliber of the organization and needs of the people.

REFERENCES

1. www.nhu.edu.tw/~chun/CS-ch13-File%20Structures.pdf
2. https://www.slideshare.net/rmmcoe/fundamental-file-processing-operations
3. https://en.wikipedia.org/wiki/Inverted_index
4. https://pdfs.semanticscholar.org