

Chapter 1

INTRODUCTION

The Cumulative Grade Point Average (CGPA) System is a stand-alone application. It provides a user-friendly, interactive Menu Driven Interface (MDI). All data is stored in files for persistence. The application uses 2 files: An Index file, to store the primary index, and, a Data file, to store records pertaining to the semester, for each semester.

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 the tree was not a single record, as in a binary tree, but a file block containing dozens, perhaps even hundreds, of records

It took 10 years until a solution emerged in the form of a *B-Tree*. Whereas AVL trees grow from the top down as records were added, B-Trees grew from the bottom up. B-Trees provided excellent access performance, but there was a cost: no longer could a file be accessed sequentially with efficiency. The problem was solved by adding a linked list structure at the bottom level of the B-Tree. The combination of a B-Tree and a sequential linked list is called a *B+ tree*.

B-Trees and B+ trees provide access times that grow in proportion to $\log_k N$, where N is the number of entries in the file and k is the number of entries indexed in a single block of the B-Tree structure. This means that B-Trees can guarantee that you can find 1 file entry among millions with only 3 or 4 trips to the disk. Further, B-Trees guarantee that as you add and delete entries, performance stays about the same.

Hashing is a good way to get what one wants with a single request, with files that do not change size greatly over time. Hashed indexes were used to provide fast access to files. But until recently, hashing did not work well with volatile, dynamic files. Extendible dynamic hashing can retrieve information with 1 or at most 2 disk accesses, no matter how big the file became.

1.1.2 About the File

When one talks about a file on disk or tape, one refers 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

A field that is a character array 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. One encounters problems when data is too long to fit into the allocated amount of space. One 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

One 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. One refers to these fields as *length-based*.

Method 3: Separate the Fields with Delimiters

One can preserve the identity of fields by separating them with delimiters. All one needs 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 one doesn't know ahead of time which fields the file is supposed to contain. It is also a good format for dealing with missing fields. If a field is missing, this format makes it obvious,

because the keyword is simply not there. It is helpful to use this in combination with delimiters, to show division between each value and the keyword for the following field. But this also wastes a lot of space: 50% or more of the file's space could be taken up by the keywords.

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

Record Structures

The five most often used methods for organizing records of a file 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 this field and record structure, one has 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, one can specify that it will contain a fixed number of fields.

Method 3: Begin Each Record with a Length Indicator

One can communicate the length of records by beginning each record with a field 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

One 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. One looks up the position of a record in the index, and then seeks 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).

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, nonvolatile capacity, is the driving force behind file structure design. Good file structure design will give us access to all the capacity without making our applications spend a lot of time waiting for the disk.

Chapter 2

SYSTEM ANALYSIS

2.1 Analysis of Application

CGPA System is used by the Student to view his/her current CGPA over the semesters completed in an undergraduate engineering course. The system is initially used to add student records containing the internal and external marks secured in the subjects in a semester, into the file corresponding to the semester. The system can then be used to search, delete, modify or display existing records of all semesters, and, to view the current CGPA of a student over the semesters he/she has completed.

2.2 Structure used to Store the Fields and Records

Storing Fields

Fixing the Length of Fields

In the CGPA System, the USN 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. The internal and external marks are integers while the SGPA is a floating-point number, each of the numbers, converted into ASCII before writing to the data file. Simple arithmetic is sufficient to recover data from the original fields.

Separating the Fields with Delimiters

The identity of fields is preserved by separating them with delimiters. The vertical bar character has been chosen, as the delimiter here.

Storing Records

Making Records a Predictable Number of Fields

In this system, there is 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. There are 18 contiguous fields and fields are recognized simply by counting the fields modulo 18.

Using an Index to Keep Track of Addresses

A B-Tree of indexes is used to keep byte offsets for each record in the original file. The byte offsets allow us to find the beginning of each successive record and compute the length of each record. The position of a record is looked up the in the B-Tree, and used seek to the record in the data file.

Placing a Delimiter at the End of Each Record

The choice of a record delimiter for the data files is the end-of-line (new-line) character (`\n`).

.

2.3 Operations Performed on a File

Insertion

The system is initially used to add student records containing the internal and external marks secured in the subjects in a semester, into the file corresponding to the semester. The SGPA is calculated from the marks, and stored along with the other fields entered, in the data file identified by the semester entered by the user. Records with duplicate USN fields are not allowed to be inserted. The length of the USN is checked to see whether it contains only 10 characters while the internal and external marks are checked to see if they are within a predefined range (0 to 20 for internal marks and 0 to 80 for external marks).

Display

The system can then be used to display existing records of all semesters. The records are displayed based on the ordering of USN maintained in the B-Tree of indexes, which here is, an ascending order. Only records with references in the B-Tree of indexes are displayed. This prevents records marked as deleted (using '\$') from being displayed. There is also an option to display the nodes of the B-Tree level-wise.

Search

The system can then be used to search for existing records of all semesters. The user is prompted for a USN, which is used as the key in searching for records in the B-Tree of indexes. The B-Tree is searched to obtain the desired starting byte address, which is then used to seek to the desired data record in any of the semester 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 all semesters. The reference to a deleted record is removed from index while the deleted record persists in the data file. A '\$' is placed in the first byte of the first field (USN) of the record, to help distinguish it from records that should be displayed. The requested record, if found, is marked for deletion, a "record deleted" message is displayed, and the reference to it is removed from the B-Tree. If absent, a "record not found" message is displayed to the user.

Modify

The reference to a deleted record is removed from index while the deleted record persists in the data file. A '\$' is placed in the first byte of the first field (USN) of the record, to help distinguish it from records that should be displayed. The requested record, if found, is marked for deletion, a

“record deleted” message is displayed, and the reference to it is removed from the B-Tree. If absent, a “record not found” message is displayed to the user. If present, after deletion, the system is used to insert the modified student record containing, possibly, a new set of internal and external marks secured in the subjects in a semester, into the file corresponding to the semester. The SGPA is calculated from the marks, and stored along with the other fields entered, in the data file identified by the semester entered by the user.

Current CGPA

The user is prompted for a USN, which is used as the key in searching for records in the B-Tree of indexes. The record with the entered USN is searched for in all semester data files. The SGPA’s retrieved from each file in which a matching record is found, are used to calculate the present CGPA of the student bearing the entered USN. A “SGPA not found”, prefixed with the semester, is returned for each semester data file in which a matching record is not found while a “SGPA=”, prefixed with the semester, is returned for each semester data file in which a matching record is found. The CGPA is an average of only the available SGPA’s.

2.4 Indexing Used

B-Tree

A B-Tree of simple indexes on the primary key is used to provide direct access to data records. Each node in the B-Tree consists of a primary key and reference pair of fields. The primary key field is the USN field while the reference field is the starting byte offset of the matching record in the data file, with one B-Tree of indexes per semester data file. Each B-Tree node can have a maximum of 4 and a minimum of 2 index entries.

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

Chapter 3

SYSTEM DESIGN

3.1 Design of the Fields and Records

The USN is declared as a character array that can hold a maximum of 10 characters. Checks are done to ensure the USN is of exactly 10 characters during input. The internal and external marks of each subject and lab are declared as integers, each having a predetermined range, only within which it is accepted during input (0 to 20 for internal marks and 0 to 80 for external marks). The SGPA field is declared as a floating-point integer whose value lies in the range 0.0 to 10.0.

Hence, a typical semester data file record can have up to 46 bytes of information to be stored, hence occupying a maximum of 65 bytes, in the data file. This includes the 18 bytes taken up by the field delimiters (|) and the 1 byte taken up by the record delimiter (\n).

The class declaration of a typical semester file record is as shown in Fig 3.1:

```
class student
{
public:
char usn[15];
int sub1ia,sub2ia,sub1ea,sub2ea,sub3ia,sub3ea,sub4ia,sub4ea,
sub5ia,sub5ea,sub6ia,sub6ea,lab1ia,lab1ea,lab2ia,lab2ea;
float sgpa;
void read();
void pack();
void unpack();
void upperto();
}s;
```

Figure 3.1 class *student*

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.

```
***** CUMULATIVE GRADE POINT AVERAGE (CGPA) *****
*****
1. Insertion
2. Display Records
3. Display B TREE
4. Deletion
5. Search
6. Modify
7. CGPA
8. Exit
Enter your choice: _
```

Figure 3.2 User 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 semester in which the record is to be inserted. Finally, the user must enter the USN, followed by the internal and external marks of the 6 subjects and 2 labs. The user is prompted for each input until the value entered meets the required criterion for each value. This means that, the user is prompted for:

- the USN, until the user enters one of exactly 10 characters.
- each internal mark, until the entered value lies in the range 0 to 20.
- Each external mark, until the entered value lies in the range 0 to 80.

After all the values are accepted, a “record inserted” message is displayed and the user is prompted to press any key to return back to the menu screen.

3.2.2 Display of a Record

If the operation desired is Display of Records, the user is required to enter 2 as his/her choice, from the menu displayed, after which a new screen is displayed. If there are no records in any file, the semester, followed by a “no records found” message is displayed. For the semester files with at least 1 record, each semester, followed by the details of each record within the file, with suitable headings, is displayed. In each case, the user is then prompted to press any key to return back to the menu screen. There is also an option for Display of the nodes of the B-Tree level-wise, for which the user is required to enter 3 as his/her choice.

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 semester, whose file from which a record is to be deleted. If there are no records in the file, a “no records to delete” message is displayed, and the user is prompted to press any key to return back to the menu screen. If there is at least 1 record in the file, the user is prompted for the USN, whose matching record is to be deleted. The USN entered is used as a key to search for a matching record. If none is found, a “record not found” message is displayed. If one is found, a “record deleted” message is displayed. In each case, the user is then prompted to press any key to return back to the menu screen.

3.2.4 Search of a Record

If the operation desired is Search, 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 semester, whose file the record is to be searched for. If there are no records in the file, a “no records to search”

message is displayed, and the user is prompted to press any key to return back to the menu screen. If there is at least 1 record in the file, the user is prompted for the USN, whose matching record is to be searched for. The USN entered is used as a key to search for a matching record. If none is found, a “record not found” message is displayed. If one is found, the details of the record, with suitable headings, are displayed. In each case, the user is then prompted to press any key to return back to the menu screen.

3.2.5 Modify of a Record

If the operation desired is Modify, the user is required to enter 6 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the semester, whose file from which a record is to be modified. The Modify operation is implemented as a deletion followed by an insertion. If there are no records in the file, a “no records to delete” message is displayed, and the user is prompted to press any key to return back to the menu screen. If there is at least 1 record in the file, the user is prompted for the USN, whose matching record is to be deleted.

The USN entered is used as a key to search for a matching record. If none is found, a “record not found” message is displayed and the user is then prompted to press any key to return back to the menu screen. If one is found, a “record deleted” message is displayed, after which a new screen is displayed. Next, the user is prompted to enter the semester in which the modified record is to be inserted. Finally, the user must enter the USN, followed by the internal and external marks of the 6 subjects and 2 labs. The user is prompted for each input until the value entered meets the required criterion for each value. This means that, the user is prompted for:

- the USN, until the user enters one of exactly 10 characters.
- each internal mark, until the entered value lies in the range 0 to 20.
- Each external mark, until the entered value lies in the range 0 to 80.

After all the values are accepted, a “record inserted” message is displayed and the user is prompted to press any key to return back to the menu screen.

3.2.6 Current CGPA

If the operation desired is CGPA, the user is required to enter 7 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, if at least one of the files contains records, the user is prompted for the USN, whose current CGPA is to be calculated and displayed. The USN entered is used as a key to search for a matching record. For each file in which a matching record is not found, the semester, followed by an “SGPA not found” message is displayed. For each file

in which one is found, the semester, followed by a “SGPA=” and the corresponding SGPA, is displayed. Then, the CGPA, from the available SGPA’s, is displayed following a “current CGPA=” message. If none of the files have any records, an additional “no entries in files” message is displayed. In each case, the user is then prompted to press any key to return back to the menu screen.

3.2.7 Design of Index

The B-Tree is declared as a class, an object of which represents a node in the B-Tree. Each node contains a count of the number of entries in the node and a reference to each descendant. The maximum number of descendants is 4 while the minimum is 2. Each node has an array of objects, each an instance of the class type *index* as shown in Figure 3.3 and Figure 3.4. Each object of type “index” contains a USN field and an address field. One object, representing the root of the B-Tree, is created for each semester file. The contents of the B-Tree are written to the index file on disk after each insert, delete and modify operation. To ensure efficient space utilization, and, to handle the conditions of underflow and overflow, nodes may be merged with their siblings, or split into 2 descendants, thereby creating a new root node for the new nodes created, or, entries within nodes maybe shifted to save space.

The above operations are used to maintain a balanced tree-structure after each insertion and deletion. The links of a node, beginning at the root, are traversed recursively, while displaying, and, writing to the index file, in order to maintain an ascending order among index entries. The same links are traversed when there is a request to search for a record, if present. The address fields of each node contain the ASCII form of the actual integer byte-offset, obtained from the data files’ get pointers. They are converted into integers, before seeking to a record in any data file, as in the case of retrieving details of a record found using a search operation. The retrieved details are either displayed, with suitable headings, as is (for Display and Search) or are used to calculate the current CGPA of a student with some USN.

```
class index
{
public:
char iusn[15],addr[5];
void initial();
void write();
};
```

Figure 3.3 class *index*

```
class btreeNode {
public:
index val[MAX + 1];
int count;
btreeNode *link[MAX + 1];
};
btreeNode *rootx,*rooty,*root;
```

Figure 3.4 class *breenode*

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. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights.

4.1.1 Classes and Objects

A semester file is declared as a *student* object with the USN, internal and external marks in 6 subjects and 2 labs, along with a SGPA, as its data members. An object of this class type is used to store the values entered by the user, for the fields represented by the above data members, to be written to the data file.

The B-Tree is declared as the class *btreenode*, an object of which represents a node in the B-Tree. Each node contains a count of the number of entries in the node and a reference to each descendant. The maximum number of descendants is 4 while the minimum is 2. Each node also has an array of objects, each an instance of the class type *index*. Each object of type *index* contains a USN field and an address field, which stores the ASCII representation of the starting byte address at which the corresponding data record is stored in the data file. Class objects are created and allocated memory only at runtime.

4.1.2 Dynamic Memory Allocation and Pointers

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

4.1.3 File Handling

Files form the core of this project and are used to provide persistent storage for user entered information on disk. The *open()* and *close()* methods, as the names suggest, are defined in the C++

Stream header file *fstream.h*, to provide mechanisms to open and close files. The *physical* file handles used to refer to the *logical* filenames, are also used to ensure files exist before use and that existing files aren't overwritten unintentionally.

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

4.1.4 Character Arrays and Character functions

Character arrays are used to store the USN fields to be written to data files and stored in a B-Tree index object. They are also used to store the ASCII representations of the integer (internal and external marks of subjects and labs) and floating-point fields (SGPA), that are written to the data file, and, the starting byte offsets of data records, to be written to the index 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

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation. The purpose of using pseudocode is that it is easier for people to understand code.

4.2.1 Insertion Module Pseudocode

The *insertion()* function (Figure 4.1) adds index objects to the B-Tree if the USN 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.

```

/* insert val in B-Tree */
void insertion(index val) {
    int flag;
    index i;
    btreeNode *child;
    // cout<<"start\n";
    flag = setValueInNode(val, &i, root, &child);
    if (flag)
        root = createNode(i, child);
}

```

Figure 4.1 *insertion()* function

4.2.2 Display Module Pseudocode

The *traversal()* function (Figure 4.2) traverses the B-Tree in ascending order accessing each index object stored at each node. The byte offsets in the objects are used to retrieve and display the corresponding data record fields. It is a recursive function.

```

/* B-Tree Traversal */
void traversal(btreeNode *myNode)
{
    int i;
    index x;
    if (myNode) {
        for (i = 0; i < myNode->count; i++) {
            traversal(myNode->link[i]);
            strcpy(x. iusn, myNode->val[i+1]. iusn);
            searching(x, &ch, root);
        }
        traversal(myNode->link[i]);
    }
}

```

Figure 4.2 *traversal()* function

There is also an option to display the nodes of a B-Tree level-wise using the *dispbtree()* function (Figure 4.3). It is a recursive function.

```

void dispbtree(btreeNode *p)
{
    int i;
    static int j=1;
    cout<<"\n Level-1->" << j++ <<"\n";
    cout<<" NODE " <<d++;
    for(i=0; i<p->count; i++)
        cout <<" ENTRY "<<i+1<<" " << p->val[i+1]. iusn;
    cout <<"\n";
    for(i=0; i<=p->count; i++)
        if (p->link[i])
        {
            dispbtree(p->link[i]);
        }
    j--;
}

```

Figure 4.3 *dispbtree()* function

4.2.3 Deletion Module Pseudocode

The *deletion()* function (Figure 4.4) deletes values from nodes and balances the B-Tree after, by merging or redistributing objects in the nodes. It makes calls to other recursive and non-recursive functions.

```

/* delete val from B-tree */
void deletion(index val, btreeNode *myNode) {
    btreeNode *tmp;
    z=1;
    if (!delValFromNode(val, myNode)) {
        x=0;
        cout<<"\n\tRecord not found\n";
        return;
    } else {
        if (myNode->count == 0) {
            tmp = myNode;
            myNode = myNode->link[0];
            free(tmp);
        }
        root = myNode;
        return;
    }
}

```

Figure 4.4 deletion() function

4.2.4 Search Module Pseudocode

The *searching()* function (Figure 4.5) traverses the B-Tree, based on the values of objects in the root node. It is a recursive function.

```

/* search val in B-Tree */
void searching(index val, int *pos, btreeNode *myNode) {
    if (!myNode) { return; }
    if (strcmp(val.iusn, myNode->val[1].iusn) < 0) {
        *pos = 0;
    } else {
        for (*pos = myNode->count;
            (strcmp(val.iusn, myNode->val[*pos].iusn) < 0 && *pos > 1); (*pos)--);
        if (strcmp(val.iusn, myNode->val[*pos].iusn) == 0) {
            h=1;
            sfile.seekg(atoi(myNode->val[*pos].addr), ios::beg);
            s.unpack();
            if (y==1)
            {
                cout<<"\n\tRecord found\n";
                cout<<endl<<setw(12)<<"USN"<<setw(8)<<"CS11"<<setw(8)<<"CS12"<<
                <<setw(8)<<"CS13"<<setw(8)<<"CS14"<<setw(8)<<"CS15"<<setw(8)<<"CS16"<<
                <<setw(8)<<"CSL1"<<setw(8)<<"CSL2"<<setw(4)<<"SGPA"<<endl;
                cout<<"          "<<setw(4)<<"In"<<setw(4)<<"Ext"<<setw(4)<<"In"<<
                <<setw(4)<<"Ext"<<setw(4)<<"In"<<setw(4)<<"Ext"<<setw(4)<<"In"<<
                <<setw(4)<<"Ext"<<setw(4)<<"In"<<setw(4)<<"Ext"<<setw(4)<<"In"<<
                <<setw(4)<<"Ext"<<setw(4)<<"In"<<setw(4)<<"Ext"<<setw(4)<<"In"<<
                <<setw(4)<<"Ext"<<endl;
            }
            cout<<setw(12)<<s.usn<<setw(4)<<s.sub1ia<<setw(4)<<s.sub2ia<<setw(4)<<
            <<s.sub3ia<<setw(4)<<s.sub4ia<<setw(4)<<s.sub5ia<<setw(4)<<s.sub6ia<<
            <<setw(4)<<s.lab1ia<<setw(4)<<s.lab2ia<<setw(4)<<s.sub1ea<<setw(4)<<
            <<s.sub2ea<<setw(4)<<s.sub3ea<<setw(4)<<s.sub4ea<<setw(4)<<s.sub5ea<<
            <<setw(4)<<s.sub6ea<<setw(4)<<s.lab1ea<<setw(4)<<s.lab2ea<<setw(4)<<
            <<s.sgpa<<endl;
            return; }
        searching(val, pos, myNode->link[*pos]);
    }
}

```

Figure 4.5 searching() function

4.2.5 Modify Module Pseudocode

The modify operation is implemented as a call to the *deletion()* function followed by a call to the *insertion()* function (Figure 4.6).

```

/* delete val from B-tree */
void deletion(index val, btreeNode *myNode) {
    btreeNode *tmp;
    z=1;
    if (!delValFromNode(val, myNode)) {
        x=0;
        cout<<"\n\tRecord not found\n";
        return;
    } else {
        if (myNode->count == 0) {
            tmp = myNode;
            myNode = myNode->link[0];
            free(tmp);
        }
        root = myNode;
        return;
    }
}

```

```

/* insert val in B-Tree */
void insertion(index val) {
    int flag;
    index i;
    btreeNode *child;
    // cout<<"start\n";
    flag = setValueInNode(val, &i, root, &child);
    if (flag)
        root = createNode(i, child);
}

```

Figure 4.6 sequence of calls to *deletion()* and *insertion* functions

4.2.6 CGPA Module Pseudocode

The *calcgpa()* is similar to the *searching()* function except that it only displays the SGPA field of a data record (Figure 4.7). The CGPA is calculated from the available SGPA's and displayed.

```

void calcgpa(index val, int *pos, btreeNode *myNode)
{
    if (!myNode) {
        return;
    }
    if (strcmp(val.iusn, myNode->val[1].iusn) < 0) {
        *pos = 0;
    } else {
        for (*pos = myNode->count;
            (strcmp(val.iusn, myNode->val[*pos].iusn) < 0 && *pos > 1); (*pos)--);
        if (strcmp(val.iusn, myNode->val[*pos].iusn) == 0) {
            f=1;
            FILE *seekg(atoi(myNode->val[*pos].addr), ios::beg);
            s.unpack();
            cout<<"SGPA = "<<s.sgpa<<endl;
            cgpa+=s.sgpa;
            return; } }
    calcgpa(val, pos, myNode->link[*pos]);
}

```

```

cgpa=0;
if(i1!=0 || i2!=0)
s.upperto();
strcpy(c.iusn,s.usn);
&sfile=&sfile1;
strcpy(datafile,datafile1);
root=rootx;
if(i1!=0 || i2!=0)
cout<<"\n\t\tFound ";
f=0;
r=i1;
if(r!=0)
{
opener(sfile,datafile,ios::in);
calcgpa(c, &ch, root);
sfile.close();
}
f1=f;
if(f1==0)
if(i1!=0 || i2!=0)
cout<<"\n\t\tFound\n";
&sfile=&sfile2;
strcpy(datafile,datafile2);
root=rooty;
r=i2;
if(i1!=0 || i2!=0)
cout<<"\n\t\tFound ";
f=0;
if(r!=0){
opener(sfile,datafile,ios::in);
calcgpa(c, &ch, root);
sfile.close();
}
f2=f;
if(f2==0)
if(i1!=0 || i2!=0)
cout<<"\n\t\tFound\n";
if((f1==0)|| (f2==0)) cgpa*=2;
cgpa/=2;
if(cgpa!=0)
cout<<"\n\t\tCurrent CGPA = "<<setprecision(2)<<cgpa;
else cout<<"\n\t\tNo entries in both files"<<endl;

```

Figure 4.7 *calcgpa()* function

4.2.7 Indexing Pseudocode

The B-Tree of indexes is written to the index file after every insertion, deletion, and modification operation, using the *write()* function (Figure 4.8), to keep index file up-to-date and consistent.

```

void write(btreeNode *myNode)
{
    int i;
    if (myNode) {
        for (i = 0; i < myNode->count; i++) {
            write(myNode->link[i]);
            ifile<<myNode->val[i + 1].iusn<<"| "<<myNode->val[i + 1].addr<<endl;
        }
        write(myNode->link[i]);
    }
}

```

Figure 4.8 *write()* function

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

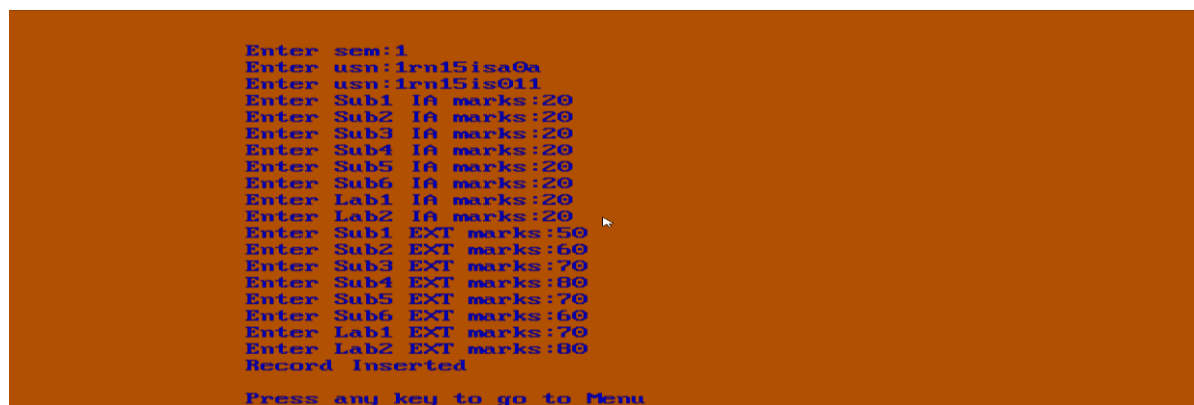
1. USN input it checked to see if it is of 10 characters of the form:

- a digit for the first character
- characters (uppercase or lowercase) for the next 2 characters

- digits for the next 2 characters
- characters (uppercase or lowercase) for the next 2 characters
- digits for the last 3 characters.

Table 4.1 Unit test case for USN Input Check

Sl No. of test case:	1
Name of test:	Check test
Item / Feature being tested:	Input for USN field
Sample Input:	USN='1RN15ISA0A'. Upon press of ENTER key.
Expected output:	Prompt for USN with 'Enter USN:' message.
Actual output:	'Enter USN:' message displayed.
Remarks:	Test succeeded

**Figure 4.9 Unit test case for USN Input Check**

2. The semester field is checked to see if it is single decimal digit.

Table 4.2 Unit test case for SEM Input Check

Sl No. of test case:	2
Name of test:	Check test
Item / Feature being tested:	Input for SEM field
Sample Input:	SEM='A'. Upon press of ENTER key.
Expected output:	Prompt for SEM with 'Enter SEM:' message.
Actual output:	'Enter SEM:' message displayed.
Remarks:	Test succeeded

```

Enter sem:a
Enter sem:2
Enter usn:1rn15is014
Enter Sub1 IA marks:20
Enter Sub2 IA marks:20
Enter Sub3 IA marks:20
Enter Sub4 IA marks:20
Enter Sub5 IA marks:20
Enter Sub6 IA marks:20
Enter Lab1 IA marks:20
Enter Lab2 IA marks:20
Enter Sub1 EXT marks:40
Enter Sub2 EXT marks:50
Enter Sub3 EXT marks:60
Enter Sub4 EXT marks:70
Enter Sub5 EXT marks:80
Enter Sub6 EXT marks:70
Enter Lab1 EXT marks:80
Enter Lab2 EXT marks:80
Record Inserted
Press any key to go to Menu

```

Figure 4.10 Unit test case for SEM Input Check

3. Each of internal and external marks fields are checked to see if they are two-digit decimal numbers within their predefined ranges (0 to 20 for internal marks and 0 to 80 for external marks).

Table 4.3 Unit test case for MARKS Input Check

Sl No. of test case:	3
Name of test:	Check test
Item / Feature being tested:	Input for MARKS fields
Sample Input:	Sub1 IA marks='AA'. Upon press of ENTER key.
Expected output:	Prompt for marks with 'Enter Sub1 IA marks:' message.
Actual output:	'Enter Sub1 IA marks:' message displayed.
Remarks:	Test succeeded

```

Enter sem:2
Enter usn:1rn15is016
Enter Sub1 IA marks:22
Enter Sub1 IA marks:aa
Enter Sub1 IA marks:20
Enter Sub2 IA marks:20
Enter Sub3 IA marks:20
Enter Sub4 IA marks:20
Enter Sub5 IA marks:20
Enter Sub6 IA marks:20
Enter Lab1 IA marks:20
Enter Lab2 IA marks:20
Enter Sub1 EXT marks:85
Enter Sub1 EXT marks:bc
Enter Sub1 EXT marks:50
Enter Sub2 EXT marks:60
Enter Sub3 EXT marks:70
Enter Sub4 EXT marks:60
Enter Sub5 EXT marks:50
Enter Sub6 EXT marks:60
Enter Lab1 EXT marks:70
Enter Lab2 EXT marks:80
Record Inserted

```

Figure 4.11 Unit test case for MARKS Input Check

4. Each file handle is used to check if files exist and if they can be opened in the required access modes or not.

Table 4.4 Unit test case for File Accesses

Sl No. of test case:	4
Name of test:	Check test
Item / Feature being tested:	File Accesses
Sample Input:	Choice 3 entered.
Expected output:	Display “No records” message for each empty file followed by “Press any key to go back to Menu” message.
Actual output:	2 “No records” messages followed by “Press any key to go back to Menu” message displayed.
Remarks:	Test succeeded

**Figure 4.12 Unit test case for File Access**

4.3.2 Integration Testing

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

Table 4.5 Integration test case for Insertion module

Sl No. of test case:	1
Name of test:	Check test
Item / Feature being tested:	Insertion module
Sample Input:	SEM=1 and USN='1RN15IS001'. Upon press of ENTER key.
Expected output:	USN and marks inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message displayed.
Actual output:	USN and marks inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message is displayed.
Remarks:	Test succeeded

```

Enter sem:1
Enter usn:1RN15IS001
Enter Sub1 IA marks:20
Enter Sub2 IA marks:20
Enter Sub3 IA marks:20
Enter Sub4 IA marks:20
Enter Sub5 IA marks:20
Enter Sub6 IA marks:20
Enter Lab1 IA marks:20
Enter Lab2 IA marks:20
Enter Sub1 EXT marks:20
Enter Sub2 EXT marks:20
Enter Sub3 EXT marks:40
Enter Sub4 EXT marks:50
Enter Sub5 EXT marks:60
Enter Sub6 EXT marks:70
Enter Lab1 EXT marks:80
Enter Lab2 EXT marks:80
Record Inserted

Press any key to go to Menu

```

Figure 4.13 Integration test case for Insertion module

Table 4.6 Integration test case for Insertion module

SI No. of test case:	2
Name of test:	Check test
Item / Feature being tested:	Insertion module
Sample Input:	SEM=1 and USN='1RN15IS001'. Upon press of ENTER key.
Expected output:	"Duplicates not allowed" message followed by "Press any key to go back to Menu" message displayed.
Actual output:	"Duplicates not allowed" message followed by "Press any key to go back to Menu" message is displayed.
Remarks:	Test succeeded

```

Enter sem:1
Enter usn:1RN15IS001
Duplicates not allowed

Press any key to go to Menu

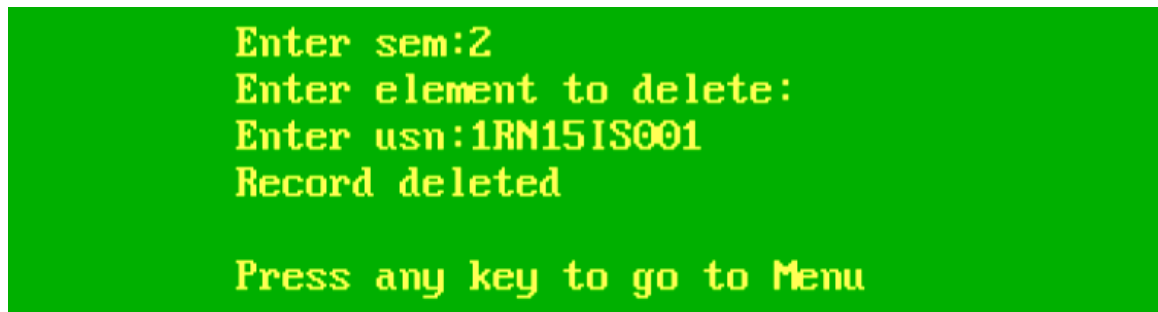
```

Figure 4.14 Integration test case for Insertion module

- 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 4.7 Integration test case for Deletion module

Sl No. of test case:	3
Name of test:	Check test
Item / Feature being tested:	Deletion module
Sample Input:	SEM=2 and USN='1RN15IS001'. Upon press of ENTER key.
Expected output:	"Record Deleted" message followed by "Press any key to go back to Menu" message displayed.
Actual output:	"Record Deleted" message followed by "Press any key to go back to Menu" message is displayed.
Remarks:	Test succeeded



```
Enter sem:2
Enter element to delete:
Enter usn:1RN15IS001
Record deleted

Press any key to go to Menu
```

Figure 4.15 Integration test case for Deletion module**Table 4.8 Integration test case for Deletion module**

Sl No. of test case:	4
Name of test:	Check test
Item / Feature being tested:	Deletion module
Sample Input:	SEM=2 and USN='1RN15IS003'. 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

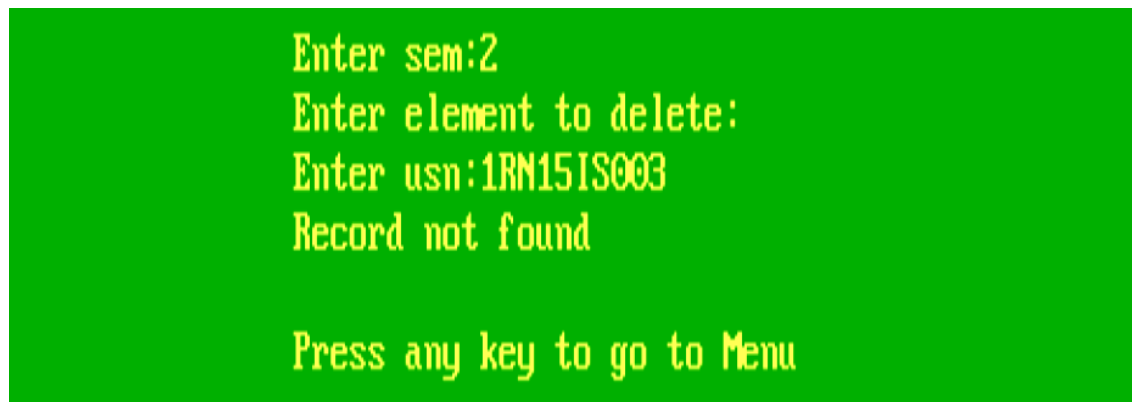


Figure 4.16 Integration test case for Deletion module

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 4.9 Integration test case for Search module

SI No. of test case:	5
Name of test:	Check test
Item / Feature being tested:	Search module
Sample Input:	SEM=1 and USN='1RN15IS004'. Upon press of ENTER key.
Expected output:	"Record found" message followed by details of the record and a "Press any key to go back to Menu" message displayed.
Actual output:	"Record found" message followed by details of the record and a "Press any key to go back to Menu" message is displayed.
Remarks:	Test succeeded

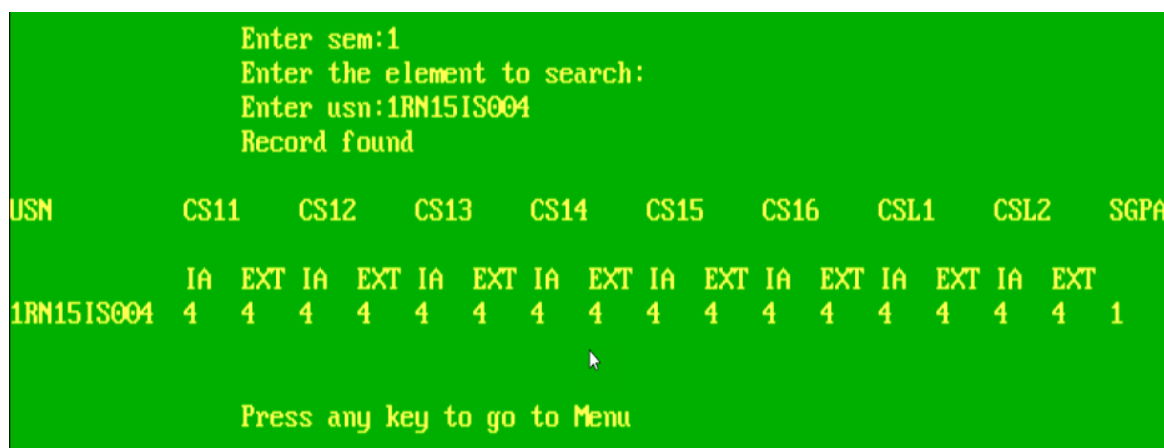
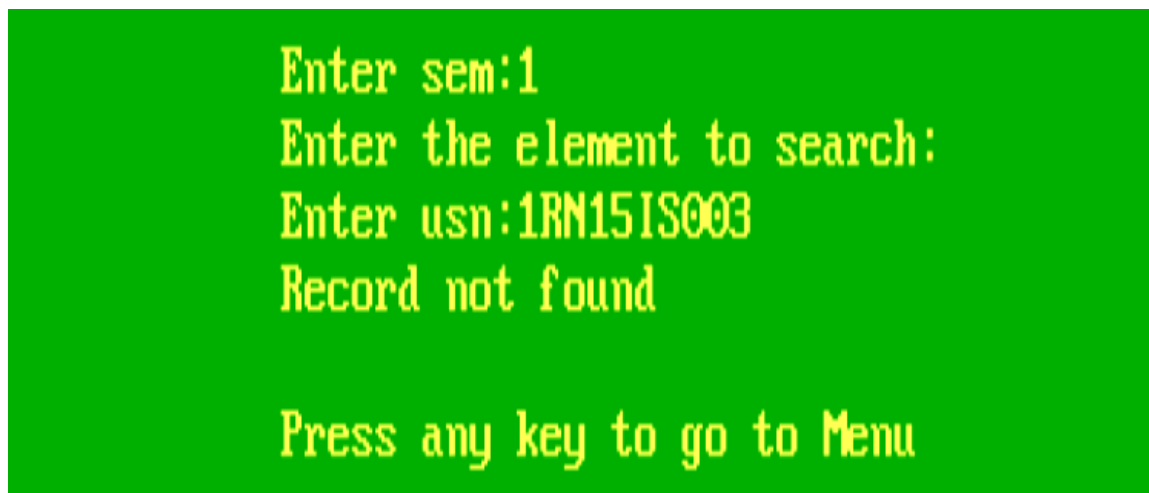


Figure 4.17 Integration test case for Search module

Table 4.10 Integration test case for Search module

Sl No. of test case:	6
Name of test:	Check test
Item / Feature being tested:	Search module
Sample Input:	SEM=1 and USN='1RN15IS003'. 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

**Figure 4.18 Integration test case for Search module**

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 4.11 Integration test case for Modify module

Sl No. of test case:	7
Name of test:	Check test
Item / Feature being tested:	Modify module
Sample Input:	SEM=1 and USN='1RN15IS004'. Upon press of ENTER key.
Expected output:	'Record deleted' message displayed. USN and marks inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message displayed.
Actual output:	'Record deleted' message is displayed. USN and marks inputs accepted, followed by 'Record inserted' message and a 'Press any key to go back to Menu' message is displayed.
Remarks:	Test succeeded

```

Enter sem:1
Enter element to delete:
Enter usn:1RN15IS004
Record deleted
Enter usn:1RN15IS005
Enter Sub1 IA marks:20
Enter Sub2 IA marks:20
Enter Sub3 IA marks:20
Enter Sub4 IA marks:20
Enter Sub5 IA marks:20
Enter Sub6 IA marks:20
Enter Lab1 IA marks:20
Enter Lab2 IA marks:20
Enter Sub1 EXT marks:40
Enter Sub2 EXT marks:50
Enter Sub3 EXT marks:60
Enter Sub4 EXT marks:50
Enter Sub5 EXT marks:60
Enter Sub6 EXT marks:70
Enter Lab1 EXT marks:80
Enter Lab2 EXT marks:80
Record Inserted

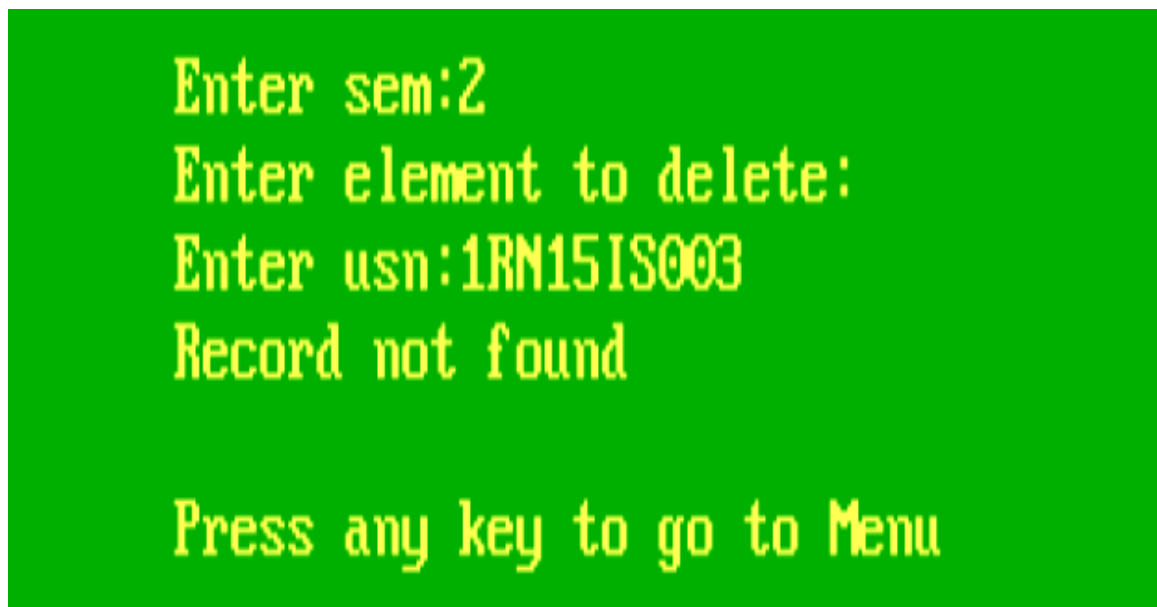
Press any key to go to Menu

```

Figure 4.19 Integration test case for Modify module

Table 4.12 Integration test case for Modify module

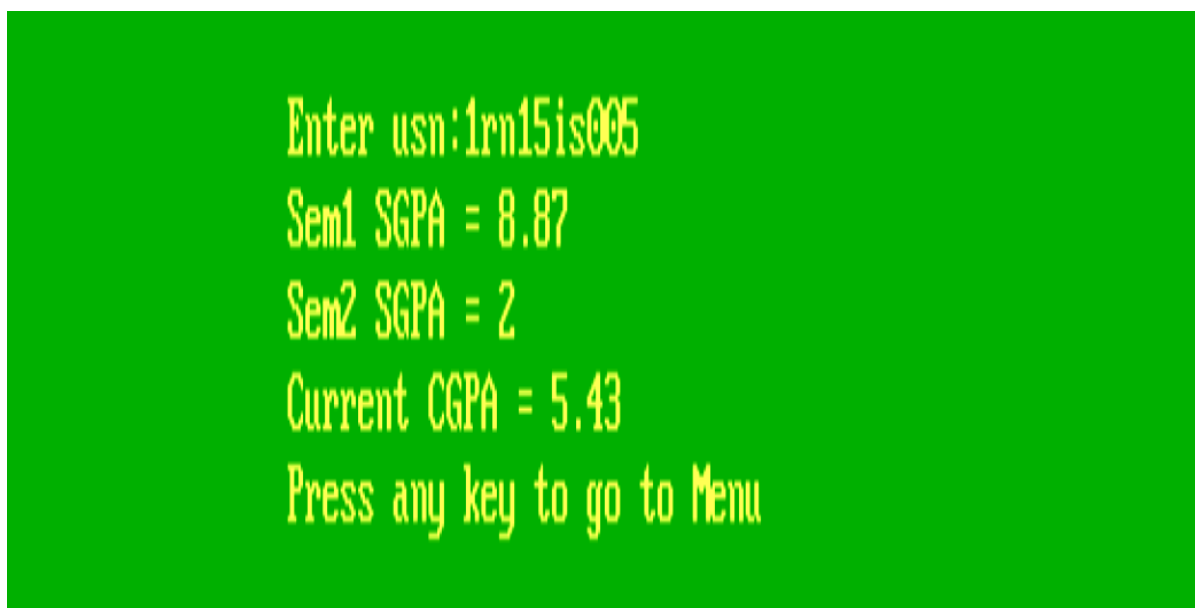
SI No. of test case:	8
Name of test:	Check test
Item / Feature being tested:	Modify module
Sample Input:	SEM=2 and USN='1RN15IS003'. Upon press of ENTER key.
Expected output:	"Record not found" message followed by "Press any key to go back to Menu" message displayed.
Actual output:	"Record not found" message followed by "Press any key to go back to Menu" message is displayed.
Remarks:	Test succeeded

**Figure 4.20 Integration test case for Modify module**

5. The CGPA function is checked to see if validation of input values is performed correctly, if correct results are returned based on whether a file contains records or not and whether desired record is present in a file or not, and that the correct calculation is performed from the available SGPA's.

Table 4.13 Integration test case for CGPA module

Sl No. of test case:	9
Name of test:	Check test
Item / Feature being tested:	CGPA module
Sample Input:	USN='1RN15IS005'. Upon press of ENTER key.
Expected output:	"SEM1 SGPA=", "SEM2 SGPA=" and "Current CGPA=" messages each with their values, followed by "Press any key to go back to Menu" message displayed.
Actual output:	"SEM1 SGPA=", "SEM2 SGPA=" and "Current CGPA=" messages each followed by their values, followed by "Press any key to go back to Menu" message is displayed.
Remarks:	Test succeeded

**Figure 4.21 Integration test case for CGPA module**

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 4.14 System test case for CGPA System

SI No. of test case:	1
Name of test:	Check test
Item / Feature being tested:	CGPA System
Sample Input:	Choices entered in the order: 1 2 4 2 5 2 6 2 7 3 8
Expected output:	Screens displayed (except menu screen) in order for: Insertion of a record Display of records before deletion of a record. Deletion of a record. Display of records after deletion of a record. Search for a record. Display of records before modification of a record. Modification of a record. Display of records after modification of a record. Display of CGPA for a USN. Display of B-Trees
Actual output:	Screens are displayed in order
Remarks:	Test succeeded

<pre> Enter sem:1 Enter usn:1RN15IS001 Enter Sub1 12 marks:20 Enter Sub2 12 marks:20 Enter Sub3 12 marks:20 Enter Sub4 12 marks:20 Enter Sub5 12 marks:20 Enter Sub6 12 marks:20 Enter Lab1 12 marks:20 Enter Lab2 12 marks:20 Enter Sub1 EXT marks:20 Enter Sub2 EXT marks:20 Enter Sub3 EXT marks:40 Enter Sub4 EXT marks:50 Enter Sub5 EXT marks:60 Enter Sub6 EXT marks:70 Enter Lab1 EXT marks:80 Enter Lab2 EXT marks:80 Record Inserted Press any key to go to Menu </pre>	<pre> Sem2: USN CS21 CS22 CS23 CS24 CS25 CS26 CSL1 CSL2 SGPA 1RN15IS001 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1RN15IS002 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1RN15IS003 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1RN15IS004 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1RN15IS005 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 1RN15IS007 7 7 7 7 7 7 7 7 7 7 7 7 7 7 2 Press any key to go to Menu </pre>
a. Insertion of a record	b. Display of records.
<pre> Enter sem:2 Enter element to delete: Enter usn:1RN15IS001 Record deleted Press any key to go to Menu </pre>	<pre> Sem2: USN CS21 CS22 CS23 CS24 CS25 CS26 CSL1 CSL2 SGPA 1RN15IS002 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1RN15IS003 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1RN15IS004 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1RN15IS005 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 1RN15IS007 7 7 7 7 7 7 7 7 7 7 7 7 7 7 2 Press any key to go to Menu </pre>
c. Deletion of a record.	d. Display of records.

<pre> Enter sem:1 Enter the element to search: Enter usn:1RN15IS004 Record found USN CS11 CS12 CS13 CS14 CS15 CS16 CSL1 CSL2 SGPA 1RN15IS004 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 Press any key to go to Menu </pre>	<pre> Sem1: USN CS11 CS12 CS13 CS14 CS15 CS16 CSL1 CSL2 SGPA 1RN15IS001 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1RN15IS002 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1RN15IS003 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1RN15IS004 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1RN15IS009 8 8 9 9 9 9 9 9 9 9 9 9 9 9 2 Press any key to go to Menu </pre>
a. Search for a record.	b. Display of records.
<pre> Enter sem:1 Enter element to delete: Enter usn:1RN15IS004 Record deleted Enter usn:1RN15IS005 Enter Sub1 IA marks:20 Enter Sub2 IA marks:20 Enter Sub3 IA marks:20 Enter Sub4 IA marks:20 Enter Sub5 IA marks:20 Enter Sub6 IA marks:20 Enter Lab1 IA marks:20 Enter Lab2 IA marks:20 Enter Sub1 EXT marks:40 Enter Sub2 EXT marks:50 Enter Sub3 EXT marks:60 Enter Sub4 EXT marks:50 Enter Sub5 EXT marks:60 Enter Sub6 EXT marks:70 Enter Lab1 EXT marks:80 Enter Lab2 EXT marks:80 Record Inserted Press any key to go to Menu </pre>	<pre> Sem1: USN CS11 CS12 CS13 CS14 CS15 CS16 CSL1 CSL2 SGPA 1RN15IS001 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1RN15IS002 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1RN15IS003 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1RN15IS005 20 20 20 20 20 20 20 20 40 50 60 50 60 70 80 80 8.87 1RN15IS009 8 8 9 9 9 9 9 9 9 9 9 9 9 9 2 Press any key to go to Menu </pre>
c. Modification of a record	d. Display of records.
<pre> Enter usn:1rn15is005 Sem1 SGPA = 8.87 Sem2 SGPA = 2 Current CGPA = 5.43 Press any key to go to Menu </pre>	<pre> Sem1 B TREE: Level1-1: NODE 1 ENTRY 1 1RN15IS003 ----- Level1-2: NODE 2 ENTRY 1 1RN15IS001 ENTRY 2 1RN15IS002 ----- Level1-2: NODE 3 ENTRY 1 1RN15IS004 ENTRY 2 1RN15IS009 ----- Sem2 B TREE: Level1-1: NODE 1 ENTRY 1 1RN15IS003 ----- Level1-2: NODE 2 ENTRY 1 1RN15IS001 ENTRY 2 1RN15IS002 ----- Level1-2: NODE 3 ENTRY 1 1RN15IS004 ENTRY 2 1RN15IS005 ENTRY 3 1RN15IS007 ----- Press any key to go to Menu </pre>
e. Display of CGPA for a USN.	f. Display of B-Trees.

Figure 4.22 (a) to (j) System test case for CGPA System

4.4 Discussion of Results

All the menu options provided in the application and its operations have been presented in as screen shots from Fig 4.23 to 4.34

4.4.1 Menu Options

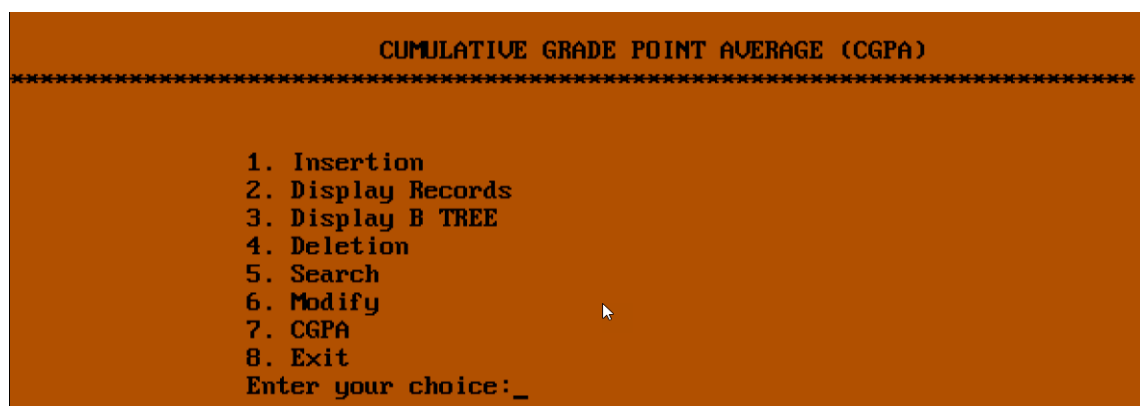


Figure 4.23 User Menu Screen

4.4.2 Insertion

```

Enter sem:1
Enter usn:1RN15IS001
Enter Sub1 IA marks:20
Enter Sub2 IA marks:20
Enter Sub3 IA marks:20
Enter Sub4 IA marks:20
Enter Sub5 IA marks:20
Enter Sub6 IA marks:20
Enter Lab1 IA marks:20
Enter Lab2 IA marks:20
Enter Sub1 EXT marks:20
Enter Sub2 EXT marks:20
Enter Sub3 EXT marks:40
Enter Sub4 EXT marks:50
Enter Sub5 EXT marks:60
Enter Sub6 EXT marks:70
Enter Lab1 EXT marks:80
Enter Lab2 EXT marks:80
Record Inserted
Press any key to go to Menu

```

Figure 4.24 Insertion of a Record.

4.4.3 Before and After Deletion

Sem2 :

USN	CS21		CS22		CS23		CS24		CS25		CS26		CSL1		CSL2		SGPA
	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	
1RN15IS001	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1RN15IS002	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1
1RN15IS003	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1
1RN15IS004	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1
1RN15IS005	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	2
1RN15IS007	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	2

Press any key to go to Menu

Figure 4.25 Display of records before deletion of a record.

```

Enter sem:2
Enter element to delete:
Enter usn:1RN15IS001
Record deleted
Press any key to go to Menu

```

Figure 4.26 Deletion of a record.

Sem2 :

USN	CS21		CS22		CS23		CS24		CS25		CS26		CSL1		CSL2		SGPA
	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	IA	EXT	
1RN15IS002	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1
1RN15IS003	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	1
1RN15IS004	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1
1RN15IS005	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	2
1RN15IS007	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	2

Press any key to go to Menu

Figure 4.27 Display of records after deletion of a record.

4.4.4 Searching a Record

```

Enter sem:1
Enter the element to search:
Enter usn:1RN15IS004
Record found

USN          CS11    CS12    CS13    CS14    CS15    CS16    CSL1    CSL2    SGPA
1RN15IS004  IA  EXT IA  EXT IA  EXT IA  EXT IA  EXT IA  EXT IA  EXT
              4    4    4    4    4    4    4    4    4    4    4    4    4    4    1

Press any key to go to Menu

```

Figure 4.28 Search for a record.

4.4.5 Before and After Modification

```

Sem1:

USN          CS11    CS12    CS13    CS14    CS15    CS16    CSL1    CSL2    SGPA
1RN15IS001  IA  EXT IA  EXT IA  EXT IA  EXT IA  EXT IA  EXT IA  EXT
              1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
1RN15IS002  2    2    2    2    2    2    2    2    2    2    2    2    2    2    1
1RN15IS003  3    3    3    3    3    3    3    3    3    3    3    3    3    3    1
1RN15IS004  4    4    4    4    4    4    4    4    4    4    4    4    4    4    1
1RN15IS009  8    8    9    9    9    9    9    9    9    9    9    9    9    9    2

Press any key to go to Menu

```

Figure 4.29 Display of records before modification of a record.

```

Enter sem:1
Enter element to delete:
Enter usn:1RN15IS004
Record deleted
Enter usn:1RN15IS005
Enter Sub1  IA marks:20
Enter Sub2  IA marks:20
Enter Sub3  IA marks:20
Enter Sub4  IA marks:20
Enter Sub5  IA marks:20
Enter Sub6  IA marks:20
Enter Lab1  IA marks:20
Enter Lab2  IA marks:20
Enter Sub1  EXT marks:40
Enter Sub2  EXT marks:50
Enter Sub3  EXT marks:60
Enter Sub4  EXT marks:50
Enter Sub5  EXT marks:60
Enter Sub6  EXT marks:70
Enter Lab1  EXT marks:80
Enter Lab2  EXT marks:80
Record Inserted

Press any key to go to Menu

```

Figure 4.30 Modification of a record.

Chapter 5

CONCLUSION AND FUTURE ENHANCEMENTS

The CGPA System focuses on providing the students, the ability to view their current CGPA. The application has successfully been designed and implemented using a B-Tree of simple indexes, to allow viewing of current CGPA over the semesters completed in an undergraduate engineering course. The application can be used to add student records containing the internal and external marks secured in the subjects in a semester, into the file corresponding to the semester.

The application can also be used to search, delete, modify and display existing records of any semester. The B-Tree of Simple Indexes has provided faster and direct access to records, utilizing memory storage efficiently. The system is tested and re-tested with varying constraints to ensure its effectiveness and provide error free functionality to the end user.

- ✓ The application can be hosted on the Web.
- ✓ The scope of the application can be extended to all 8 semesters of the engineering course.

REFERENCES

1. File Structures: An Object-Oriented Approach in C++, PEARSON, 3rd Edition.
2. K.R. Venugopal, K.G. Srinivas, P.M. Krishnaraj: File Structures Using C++, Tata McGraw-Hill, 2008.
3. Scot Robert Ladd: C++ Components and Algorithms, BPB Publications, 1993
4. Raghu Ramakrishnan and Johannes Gehrke: Database Management Systems, 3rd Edition, McGraw-Hill, 2003
5. www.geeksforgeeks.org.
6. uxmankabir.wordpress.com.