

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JnanaSangama, Belagavi – 590014.



MINI PROJECT REPORT

ON

“Tic Tac Toe”

Submitted in partial fulfillment for the requirement of 6th semester for the

Degree of Bachelor of Engineering in

INFORMATION SCIENCE & ENGINEERING

For the academic year 2020-21

SUBMITTED BY:

POOJITH S

[1DB18IS058]

Under the guidance of:

Prof. HARIPRIYA.C.,

Assistant Professor,

Dept. of ISE



DON BOSCO INSTITUTE OF TECHNOLOGY, BENGALURU-560074

DON BOSCO INSTITUTE OF TECHNOLOGY



Kumbalagodu, Bengaluru-560074

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project Report entitled **“Tic Tac Toe”** is a bonafide Project work carried out by **POOJITH S (1DB18IS058)**, in partial fulfillment of ‘6th’ semester for the Degree of **Bachelor of Engineering in Information Science and Engineering** of Visvesvaraya Technological University, Belagavi, during the academic year 2020-21. It is certified that all corrections/suggestions indicated for Internal Assessments have been incorporated with the degree mentioned.

Project Guide

Prof. Haripriya.C

Assistant. Prof.

Dept. of ISE,

DBIT, Bangalore.

Head of Department

Prof. Gowramma G.S

Head of Department

Dept. of ISE,

DBIT, Bangalore.

External Viva

Name of the Examiners

1. _____

2. _____

Signature with Date

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru -560074



DECLARATION

POOJITH, student of sixth semester B.E, Department of Information Science and Engineering, Don Bosco Institute of Technology, Kumbalagodu, Bengaluru, declare, that Mini Project Work entitled “**Tic Tac Toe**” has been carried out by and submitted in partial fulfillment of the requirement of 6th semester April 2021-Aug 2021. The matter embodied in this report has been submitted to any university or institute for the award of any other degree or diploma.

Place: Bengaluru

POOJITH S

Date:

(1DB18IS058)

ACKNOWLEDGEMENT

At the various stages in making the mini project, a number of people have given me invaluable comment on the manuscript. I take this opportunity to express my deepest gratitude and appreciation to all those who helped me directly or indirectly towards the successful completion of this mini project.

I would like to thank **Principal Dr HEMADRI NAIDU. T, DonBosco Institute of Technology** for his support though out this mini project.

I express my whole hearted gratitude to **Prof.GOWRAMMA G.S**, who is our respectable **Head of Dept. Department of Information Science and Engineering**. I wish to acknowledge for her valuable help and encouragement.

In this regard, a heartfelt gratitude to guide **Prof. HARIPRIYA.C. Assistant Professor, Department of Information Science and Engineering**, for her timely advice on the miniproject and regular assistance throughout the miniproject work. I would also like to thank the staff members of Department of **Information Science and Engineering** for their corporation.

ABSTRACT

The main aim and objective was to plan and program system application and to get rid of manual entry and to store it in a file so that easily available. We have to apply the best software engineering practice for system application. I developed “Tic Tac Toe game ” where the software Visual Studio for C++ is used entertain the players by playing the tic tac toe game and perform basic operations like insertion, deletion and display can be performed. The System can handle all the details about the players. The user can collect players information by adding, removing and searching for details. This mini project contains limited features, but entertain one. This details include players name, time to win and number of games played . It tracks all the details of a player from the game one to the end of his game. The summary provided will give a detailed information of the game played, it will be easy to analyze the game later.

.

CONTENTS

1. INTRODUCTION.

1.1 PROBLEM STATEMENT

1.2 REQUIREMENTS

1.3 SCOPE

2. FILE STRUCTURE INTRODUCTION

2.1 FILE STRUCTURE USED IN PROJECT

2.2 FILE FUNCTIONS

3. PSEUDOCODE

3.1 INSERTION STATEMENTS

3.2 DELETE STATEMENTS

3.3 DISPLAY STATEMENTS

4. SNAPSHOTS.

4.1 INSERTION OPERATION

4.2 DISPLAY OPERATION

4.3 DELETE OPERATION

4.4 INDEXES FILES

5. ADVANTAGES OF USING FILES STRUCTURES

6. CONCLUSION.

7. REFERENCES.

Chapter-1**INTRODUCTION**

I have developed Tic Tac Toe System to get rid from manual entry and store it in a file which can be easily available. Here we perform various operations like inserting a record, deleting a record, and displaying a record. For this purpose, need the software which could easily and conveniently maintain the players details and to entertain them. The record of player can be stored in a single file. This project “Tic Tac Toe management system” includes some facilities such as players name, winners name, time to win etc .

1.1 PROBLEM STATEMENT

The main aim of designing this project is to get rid from manual entry and store it in a file which is easily available and perform operations like insert, delete, display the records for player record management system.

1.2 REQUIREMENTS**HARDWARE REQUIREMENT-**

Minimum RAM :- 2GB

Processor :- Intel Pentium 5

Operating System :- Windows 10

SOFTWARE REQUIREMENT-.

Language :- C++

Software used-Visual studio for C++

1.3 Scope

The software product “**Tic Tac Toe**” will be an application that will be Used for maintaining the records in an organized manner and to replace old paper work system. This project aims at automating the player details for smooth working of the database by automating almost all the activities.

Chapter-2.

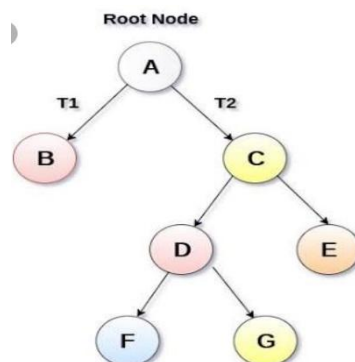
FILE STRUCTURE INTRODUCTION

File Structures is the Organization of Data in Secondary Storage Device in such a way that minimize the access time and the storage space. 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.

SHORT HISTORY OF FILE STRUCTURE DESIGN:

General goals of research and development in file structures:

- To find the target information with as few access as possible (i.e., 2 or 3 accesses).
- To get the information we need with one access to the disk.
- To group information to get everything with only one access.
- Trees -in the early 1960s, the idea of applying tree structures emerged as a potential solution. Unfortunately, trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record.



- Sequential access -> Direct access •

Tape -> Disk.

Index : <key, pointer> in smaller file.

- As files grew intolerably large for unaided sequential access and as storage devices such as disk drives became available, indexes are added to files. Indexes made it

possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With the key and pointer, user had direct access to large primary file.

- AVL tree is a self-adjusting binary tree for data in memory. Other researchers began to look for ways to apply AVL trees, or something like them to files.

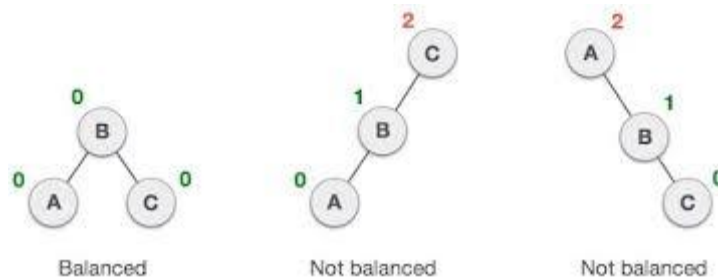


Fig: Examples for AVL tree

- It took nearly 10 years of design work before a solution emerged in the form of the B-tree. B-tree is a balanced tree structure and provide excellent access performance, but sequential access with a cost.
- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3

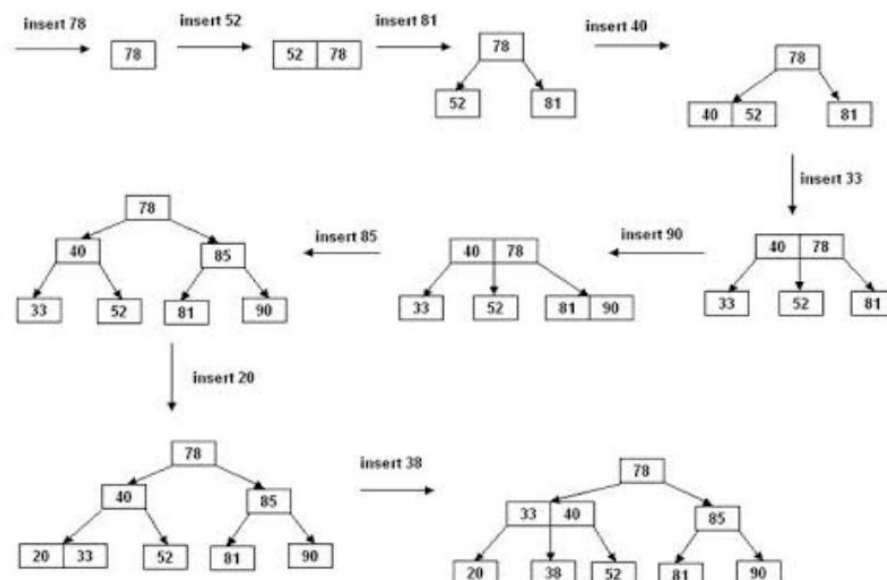


Fig: Example of B-trees

- The combination of a B-tree and sequential linked list is called a B+ tree. Over the next 10 years, B-trees and B+ trees became the basis for many commercial file systems.

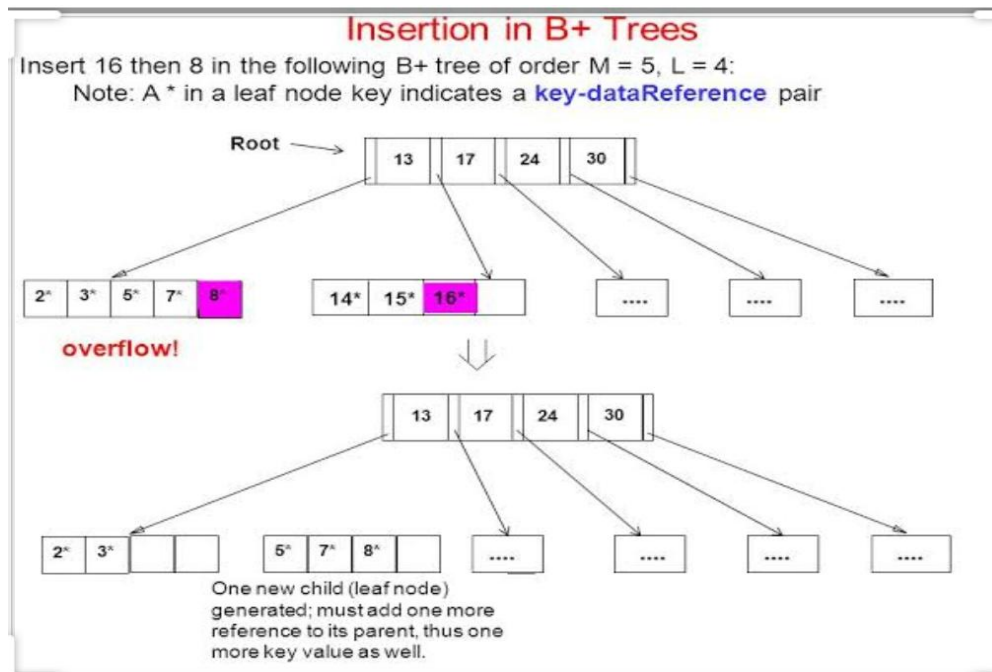


Fig: Example of B+ trees

- An approach called hashing is a good way to do that with the files that do not change size greatly over time. From early on, hashed indexes were used to provide fast access to files.
- After the development of B-trees, researchers turned to work on systems for extendible, dynamic hashing that could retrieve information with one ,or at most ,two disk access no matter how big the file became.

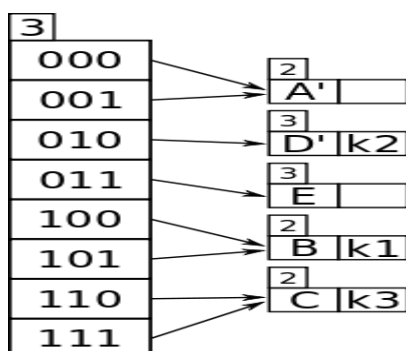


Fig:Extendible hashing

2.1 FILE STRUCTURE USED IN PROJECT:INDEXES

INDEXES:

All indexes are based on same basic concepts-keys and reference fields. The types of indexes we see are called simple indexes because they are represented using simple arrays of structures that contain the keys and reference fields.

A Simple Index for Entry-Sequenced Files

Simple index: An index in which the entries are a key ordered linear list.

- Simple indexing can be useful when the entire index can be held in memory.
- Changes (additions and deletions) require both the index and the data file to be changed.
- Updates affect the index if the key field is changed, or if the record is moved.
- An update which moves a record can be handled as a deletion followed by an addition.

Object Oriented Support for Indexed, Entry Sequenced Files

Entry-sequenced file: A file in which the record order is determined by the order in which they are entered. The index should be read into memory when the data file is opened.

- Indexes That are too Large to Hold in Memory
- Searching of a simple index on disk takes too much time.
- Maintaining a simple index on disk in sorted order takes too much time.
- Tree structured indexes such as B-trees are a scalable alternative to simple indexes.
- Hashed organization is an alternative to indexing when only a primary index is needed.

Indexing to Provide Access by Multiple Keys

Secondary key: A search key other than the primary key.

Secondary index: An index built on a secondary key.

- Secondary indexes can be built on any field of the data file, or on combinations of fields.
- Secondary indexes will typically have multiple locations for a single key.
- Changes to the data may now affect multiple indexes.
- The reference field of a secondary index can be a direct reference to the location of the entry in the data file.
- The reference field of a secondary index can also be an indirect reference to the location of the entry in the data file, through the primary key.
- Indirect secondary key references simplify updating of the file set.
- Indirect secondary key references increase access time.
- We could build a catalog for our record collection consisting of entries for album title, composer, and artist. These fields are secondary key fields . Just as the library catalog relates an author entry (secondary key) to a card catalog number (primary key) , so can we build an index file that relates Composer to Label ID, as illustrated in Fig 2.2.1.
- Along with the similarities, there is an important difference between this kind of secondary key index and the card catalog in a library. In a library, once you have the catalog number you can usually go directly to the stacks to find the book since the books are arranged in order by catalog number. In other words, the books are sorted by primary key. The actual data records in our file, on the other hand, are entry sequenced. Consequently, after consulting the composer index to find the Label ID, you must consult one additional index, our primary key index, to find the actual byte offset of the record that has this particular Label Id.
- The procedure is summarized in Fig. 2.2.2. Clearly it is possible to relate secondary key references (e.g., Beethoven) directly to a byte offset (211) rather than to a primary key (DG1 8807). However, there are excellent reasons for postponing this binding of a secondary key to a specific address for as long as possible. These reasons become clear as we discuss the way that fundamental file operations such as record deletion and updating are affected by the use of secondary indexes. Record Addition When a

secondary index is present, adding a record to the file means adding a record to the secondary index.

Composer index

Secondary key	Primary key
BEETHOVEN	ANG3795
BEETHOVEN	DG139201
BEETHOVEN	DG18807
BEETHOVEN	RCA2626
COREA	WAR23699
DVORAK	COL31809
PROKOFIEV	LON2312
RIMSKY-KORSAKOV	MER750 16
SPRINGSTEEN	COL38358
SWEET HONEY IN THE R	FF245

Fig 2.2.1 Secondary key index organized by composer.

PROCEDURE search_on_secondary (KEY)

search for KEY in the secondary index

once the correct secondary index record is found , set LABEL_ID to the primary key value in the record's reference field

call retrieve_record (LABEL_ID) to get the data record

end **PROCEDURE**

Fig 2.2.2: Search_on_secondary: an algorithm to retrieve a single record from Datafile through a secondary key index .

Retrieval Using Combinations of Secondary Keys

- The search for records by multiple keys can be done on multiple index, with the combination of index entries defining the records matching the key combination.

- If two keys are to be combined, a list of entries from each key index is retrieved.
- For an "or" combination of keys, the lists are merged.
- I.e., any entry found in either list matches the search.
- For an "and" combination of keys, the lists are matched.
- I.e., only entries found in both lists match the search.

Improving the Secondary Index Structure: Inverted Lists

Inverted list: An index in which the reference field is the head pointer of a linked list of reference items.

Selective Indexes: An index which contains keys for only part of the records in a data file.

Title index

Secondary key	Primary key
COQ D'OR SUITE	MER75016
GOOD NEWS	FF245
NEBRASKA	COL38358
QUARTET IN C SHARP	M RCA2626
ROMEO AND JULIET	LON2312
SYMPHONY NO. 9	ANG3795
SYMPHONY NO. 9	COL3 1809
SYMPHONY NO. 9	DG18807
TOUCHSTONE	WAR23699
VIOLIN CONCERTO	DG139201

Fig 2.2.3 Secondary key index organized by recording title.

The secondary index structures that we have developed so far result in two distinct difficulties :

1. We have to rearrange the index file every time a new record is added to the file, even if the new record is for an existing secondary key.

For example, if we add another recording of Beethoven's Symphony No. 9 to our collection, both the composer and title indexes would have to be rearranged, even though both indexes already contain entries for secondary keys (but not the Label IDs) that are being added.

2. If there are duplicate secondary keys, the secondary key field is repeated for each entry. This wastes space, making the files larger than necessary. Larger index files are less likely to be able to fit in electronic memory.

2.2 FILES FUNCTIONS USED IN PROJECT:

File handling :-

Many real-life problem handle large volume of data and ,in such situations, we need to use some devices such as Floppy disk or hard disk to store the data. The data is stored in these devices using the concept of files. A file is a collection of related data stored in a particular area on the disk. Programs can be designed to perform the read and write operations on these files.

Class for file stream operation :-

The I/O system of CPP contain a set of classes that define the file handling methods. This include ifstream, ofstream and fstream. These classes are derived from fstream base and the corresponding istream class, the classes ,designed manage the disk ,are declared in fstream and therefore we must include this file in any program that uses file.

Opening file Syntax:-

File stream-class stream object;

Stream object .open(“filename,ios::mode);

Mode for opening the file:-

- 1) **ios ::app** ->To append at the end of file
- 2)**ios::out**->it open file in write only mode.
- 3) **ios::in**->it open file in read only mode.

Read file syntax:-

Stream -object .read((char*)&class-object,sizeof(class-object));

Write files syntax:

Stream -object.write((char*)&class-object,sizeof(class -object));

- **f.close():-** It closes the stream. All buffers associated with the stream are flushed before closing. System allocates buffers are freed upon closing. Buffers are assigned with setbuf are not automatically freed.

Chapter-3

PSEUDOCODE

3.1 INSERTION STATEMENTS:-

MODE 1:

```
cout<<"\tEnter Player one name : ";  
  
cin>>p1;  
  
cout<<"\tEnter Player two name : ";  
  
cin>>p2;  
  
o1.open("playerlist.txt",ios::out | ios::app);  
  
o1<<"\n"<<p1<<"|"<<p2<<"|";  
  
o1.close();
```

MODE 2:

```
cout<<"\tEnter Player one name : ";  
  
cin>>p1;  
  
o1.open("playerlist.txt",ios::out | ios::app);  
  
o1<<"\n"<<p1<<"|"<<"computer"<<"|";  
  
o1.close();
```

3.2 DELETE STATEMENTS:-

```
o4.open("winnersdata.txt",ios::in);  
  
temp.open("temp.txt",ios::out | ios::app);
```

```
cout<<"      Enter the winner's name to delet from the winnersdata : ";

cin>>dname;

while(getline(o4,line))

{

    position=line.find(dname);

    if(position!=0)

    {

        temp<<line<<endl;

    }

}

o4.close();

temp.close();

o4.open("winnersdata.txt",ios::in);

while(getline(o4,line))

{

    w++;

}

w--;

temp.open("temp.txt",ios::in);

while(getline(temp,line))

{

    t++;

}

}
```

```
t--;

if(w==t)

{

    cout<<"\n!*!*!no record found!*!*!"<<endl;

}

else

{

    cout<<"\n!*!*!*!*!*!record delet successfully!*!*!*!*!*!*!"<<endl;

}

o4.close();

temp.close();

remove("winnersdata.txt");

rename("temp.txt","winnersdata.txt");
```

3.3 DISPLAY STATEMENTS:-

```
o4.open("winnersdata.txt",ios::in);

temp.open("temp.txt",ios::out|ios::app);

while(getline(o4,line))

{

    if(!line.empty())

    {

        temp<<line<<endl;

    }

}
```

```
o4.close();

temp.close();

remove("winnersdata.txt");

rename("temp.txt","winnersdata.txt");

o4.open("winnersdata.txt",ios::in);

int game=1,c=0;

while(getline(o4,line))

{

c++;

}

o4.close();

o4.open("winnersdata.txt",ios::in);

cout<<endl<<"*!*!*!*!*!* Game History *!*!*!*!*!*!*!*!"<<endl;

while(!o4.eof() && game<=c)

{

o4.getline(wname,100,'|');

o4.getline(lname,100,'|');

o4.getline(wt,100,'|');

cout<<endl<<"game "<<game<<": "<<wname<<" won against "<<lname<<" in "<<wt<<"
seconds"<<endl;

game++;

}

o4.close();

break;
```


Chapter-4

SNAPSHOTS

4.1 INSERTION OPERATION:-


There is two modes in the game: one to play with friends and to play against computer, If you select first mode two players name needs to be inserted else only one player name to inserted since other player is computer.

Mode 1: Two players

 C:\Users\Sharath\Documents\fs_project\submission\tic_tac_toe_game_with_oop.exe

```
Enter Player one name : albatross
Enter Player two name : prophet_
```


Mode 2: single player

 C:\Users\Sharath\Documents\fs_project\submission\tic_tac_toe_game_with_oop.exe

```
Enter Player name : Prarthana
```

4.3 DISPLAY OPERATION:-

The display operation is performed to display the game history in which no of games played, winner's name, loser's name and time to win is displayed.

 C:\Users\Sharath\Documents\fs_project\submission\tic_tac_toe_game_with_oop.exe


```
*!*!*!*!*!*!* Game History *!*!*!*!*!*!*!*!*!*!*!*  
  
game 1:winner won against looser in time seconds  
  
game 2:  
poojith won against vikas in 5 seconds  
  
game 3:  
poojith won against x in 11 seconds  
  
game 4:  
harsha won against bhuvi in 7 seconds  
  
game 5:  
bhuvi won against computer in 13 seconds  
  
game 6:  
albatross won against prophet in 18 seconds  
  
game 7:  
dilip won against computer in 21 seconds
```

```
albatross is the winner(imported from winnersdata.txt)  
  
-----This game summary-----  
  
    Player 1 --> albatross  
  
    Player 2 --> prophet  
  
    albatross you took 18 seconds to beat prophet
```

4.5 DELETE OPERATION:-


Delete operation includes deleting a record with the winners name. If the given winner name matches, then the record would be deleted otherwise it displays record not found and delete operation is unsuccessful.

On unsuccessful operation

 C:\Users\Sharath\Documents\fs_project\submission\tic_tac_toe_game_with_oop.exe

```
Enter the winner's name to delet from the winnersdata : 1  
!***no record found!***!
```

On successful deletion

 C:\Users\Sharath\Documents\fs_project\submission\tic_tac_toe_game_with_oop.exe

```
Enter the winner's name to delet from the winnersdata : dilip  
*****!record delet successfully*****!
```

4.6 INDEXES FILES

1: This file contains player1 name and player2 name and other attributes are included.



playerlist - Notepad

File Edit Format View Help

```
player1|player2|  
prarthana|computer|  
poojith|computer|  
vikas|computer|  
su|computer|
```

2: This file contains winner's name loser's name and time to win details which includes all the attributes in a sorted manner.



winnersdata - Notepad

File Edit Format View Help

```
winner|looser|time|  
poojith|vikas|5|  
poojith|x|11|  
harsha|bhuvi|7|  
bhuvi|computer|13|  
albatross|prophet|18|
```


Chapter-5

ADVANTAGES OF FILE STRUCTURES

Advantages of ordered File Organization:

- To find a record in the sequential file is very efficient, because all files are stored in an order.
- It is fast and efficient when dealing with large volumes of data that need to be processed periodically.

Advantages of unordered File Organization:

- insert simple, records added at end of file.
- easier for retrievals a large proportion of records.
- effective for bulk loading data.

Advantages of hashing File Organization:

- Direct Access to the data.
- Hash function or randomizing function.
- Best if equality search is needed on hash-key.

Chapter-6**CONCLUSION**

The Mini Project “**Tic Tac Toe**” is designed in order reduce the burden of maintaining bulk records of all the details in which Inserting, Retrieving and deleting the details are easy when compared to the manual update and storing. This mini project helps in maintaining the players details in an Organized manner and to replace old paper work system. This is to conclude that the project that I undertook was worked upon sincere effort. Most of the requirement are fulfilled up to the mark.

Chapter-7

REFERENCES

1. Reference Book :- Michael J. Folk, Bill Zoellick - File Structures

(1991, Addison Wesley)

2. Website:-

<http://www.cppforschool.com/projects.html>

<http://www.codeincodeblock.com/mini-projects.html>

<https://www.codewithc.com/c-projects-with-source-code/>