UNIT VI

File Organization

Files : Need

- The prime role of computers is problem solving and data processing.
- In any computer application, the basic entity is data.
- Data can be either simple or it may have multiple attributes.
- One needs to select the appropriate data structure based on the nature of the application and data.
- Data can have one or more attributes (fields).
- If you have to enter a large number of data, it will take a lot of time to enter them all.
- You can easily move your data from one computer to another without any changes.
- Without a file system, the storage device would contain a big chunk of data stored back to back, and the operating system wouldn't be able to tell them apart.

Files : Concept

- A file is a collection of records where each record consists of one or more fields.
- Records that hold information about similar items of data are usually grouped together into a file.
- For example, a file Student can have one or more records with fields such as Roll_No, Name, DOB, City, and Sex.
- Table indicates the fields and their associated data type for this record.
- Each record contains attributes to describe one entity.
- Generally, all records for one entity type are usually of the same form.
- Mostly, each of them has the same fields in the same quantity, order, and length.
- Such records are known as fixed length records. Structures in C/C++ support this type of record.
- Records that are not necessarily of the same length are known as variable length records. C/C++ unions support this type of record.

Field	Roll_No	Name	DOB	City	Sex
Datatype	Integer	Array of characters	Array of characters	Array of characters	Character

A few features of using files

- **Reusability:** Data stored in files can be used repeatedly.
- **Portability:** Files enable data transfer between different systems.
- **Efficient:** Quick access to stored data.
- **Storage Capacity:** Large amounts of data can be stored beyond the constraints of RAM.

Types of Files

- A file can be classified into two types based on the way the file stores the data.
  1. **Text Files**
  2. **Binary Files**

1. Text files

- Text files are the normal .txt files. You can easily create text files using any simple text editors .
- When you open those files, you'll see all the contents within the file as plain text. You can easily (ASCII characters) edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

2. Binary files

- Binary files are mostly the .bin files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold a higher amount of data, are not readable easily, and provides better security than text files.

Types of Files

There are a large number of file types. Each has a particular purpose and extensions. The type of a file indicates its use cases, contents, etc. Some common types are:

1. **Media :** Media files store media data such as images, audio, icons, video, etc. Common extensions: img, mp3, mp4, jpg, png, flac, etc.
2. **Programs :** These files store code, markup, commands, scripts, and are usually executable. Common extensions: c, cpp, java, xml, html, css, js, ts, py, sql, etc.
3. **Operating System Level :** These files are present with the OS for its internal use. Common extensions: bin, sh, bat, dl, etc.
4. **Document :** These files are used for managing office programs such as documents, spreadsheets, etc. Common extensions: xl, doc, docx, pdf, ppt, etc.
5. **Miscellaneous :** Generic text file(.txt), canvas files, proprietary files, etc.

Files: Primitive operations

- C++ provides us with the following operations in File Handling:
  1. Creating a file: open()
  2. Reading data: read()
  3. Writing new data: write()
  4. Closing a file: close()
- A stream is an abstraction that represents a device on which operations of input and output are performed.
- A stream can be represented as a source or destination of characters of indefinite length depending on its usage.
- In C++ we have a set of file handling methods. These include **ifstream, ofstream, and fstream.** These classes are derived from fstrembase and from the corresponding iostream class.

Classes for File Stream in C++:

- **ifstream:** Used for input file stream. This class is designed to read information from files.
- **Example:**

```
ifstream inFile;
inFile.open("sample.txt");
```

- **ofstream:** Used for output file stream. This class is used for creating files and writing information to them.
- **Example:**

```
ofstream outFile;
outFile.open("example.txt");
```

- **fstream:** Represents the generic file stream class. It can be used for both reading from and writing to files. It is more versatile as compared to ifstream and ofstream, but knowing when to use the specialized classes is crucial for effective programming.
- **Example:**

```
fstream ioFile;
ioFile.open("data.txt", ios::in | ios::out);  // Open for input and output
```

Opening a File

- To start working with the file, first, we need to open it in our program.
- We can either open our file with the constructor provided by the file I/O classes or call the open method on the stream object.
- There are several opening modes.

Opening Modes

Mode	Syntax	Description
Read	ios::in	Opens file for reading purpose.
Write	ios::out	Opens a file for writing purposes.
Binary	ios::binary	All operations will be performed in binary mode.
Truncate before Open	ios::trunc	If the file already exists, all content will be removed immediately.
Append	ios::app	All provided data will be appended in the associated file.
At End	ios::ate	It opens the file and moves the read/write control at the End of the File. The basic difference between the ios::app and this one is that the former will always start writing from the end, but we can seek any particular position with this one.

Opening a File

1. Open a File Using Constructor

- Each class has two types of constructors: default and those that specify the opening mode and the associated file for that stream.
  - **ifstream Stream_Object(const char* filename, ios_base::openmode = ios_base::in);**
  - **ofstream Stream_Object(const char* filename, ios_base::openmode = ios_base::out);**
  - **fstream Stream_Object(const char* filename, ios_base::openmode mode = ios_base::in | ios_base::out);**

Opening a File

2. Open a File Using stream.open() Method

- The open() is a public member function of all these classes. Its syntax is shown below.
  - **void open (const char* filename, ios_base::openmode mode);**
- The open() method takes two arguments one is the file name, and the other is the mode in which the file is going to open.
- The is_open() method is used to check whether the stream is associated with a file or not. It returns true if the stream is associated with some file; otherwise returns false.
  - **bool is_open();**

Reading from a File

- We read the data of a file stored on the disk through a stream. The following steps must be followed before reading a file,
  1. **Create a file stream object capable of reading a file, such as an object of ifstream or fstream class.**

```
ifstream streamObject;
// Or
fstream streamObject;
```

  2. **Open a file through constructor while creating a stream object or by calling the open method with a stream object.**

```
ifstream streamObject("myFile.txt");
// Or
streamObject.open("myFile.txt");
// Note:- If a stream is already associated with some file, then the call to open method will fail.
```

  3. **Check whether the file has been successfully opened using is_open(). If yes, then start reading.**

```
if(streamObject.is_open()){
   // File Opened successfully.
}
```

Reading from a File - Using get() Method

```
#include <fstream>
#include<iostream>

int main ()
{
    std::ifstream myfile("sample.txt");
    if (myfile.is_open()) {
        char mychar;
        while (myfile.good()) {
            mychar = myfile.get();
            std::cout << mychar;
        }
    }
    return 0;
}
```

Output:

Hi, this file contains some content.
This is the second line.
This is the last line.

Reading from a File - Using getline() Method

```
#include <fstream>
#include<iostream>
#include<string>

int main ()
{
    std::ifstream myfile("sample.txt");
    if (myfile.is_open()) {
        std::string myline;
        while (myfile.good()) {
            std::getline (myfile, myline);
            std::cout << myline << std::endl;
        }
    }
    return 0;
}
```

Output:

Hi, this file contains some content.
This is the second line.
This is the last line.

Writing to a File

- In writing, we access a file on disk through the output stream and then provide some sequence of characters to be written in the file. The steps listed below need to be followed in writing a file.

1. **Create a file stream object capable of writing a file, such as an object of ofstream or fstream class.**

```
ofstream streamObject;
// Or
fstream streamObject;
```

2. **Open a file through constructor while creating a stream object or by calling the open method with a stream object.**

```
ofstream streamObject("myFile.txt");
// Or
streamObject.open("myFile.txt");
```

3. Check whether the file has been successfully opened. If yes, then start writing.

```
if(streamObject.is_open())
{
   // File Opened successfully.
}
```

1. Writing in Normal Write Mode

```
#include <fstream>
#include<iostream>
#include<string>
int main ()
{
    // By default, it will be opened in normal write mode,
// which is ios::out.
    std::ofstream myfile("sample.txt");
    myfile << "Hello Everyone \n"; //insertion operator
    myfile << "This content was being written from a C++ Program";
    return 0;
}
```

2. Writing in Append Mode

```
#include <fstream>
#include<iostream>
#include<string>
int main ()
{
    std::ofstream myfile("sample.txt", std::ios_base::app);
    myfile << "\nThis content was appended in the File.";
    return 0;
}
```

3. Writing in Truncate Mode

```
#include <fstream>
#include<iostream>
#include<string>
int main ()
{
    std::ofstream myfile("sample.txt", std::ios_base::trunc);
    myfile << "Only this line will appear in the file.";
    return 0;
}
```

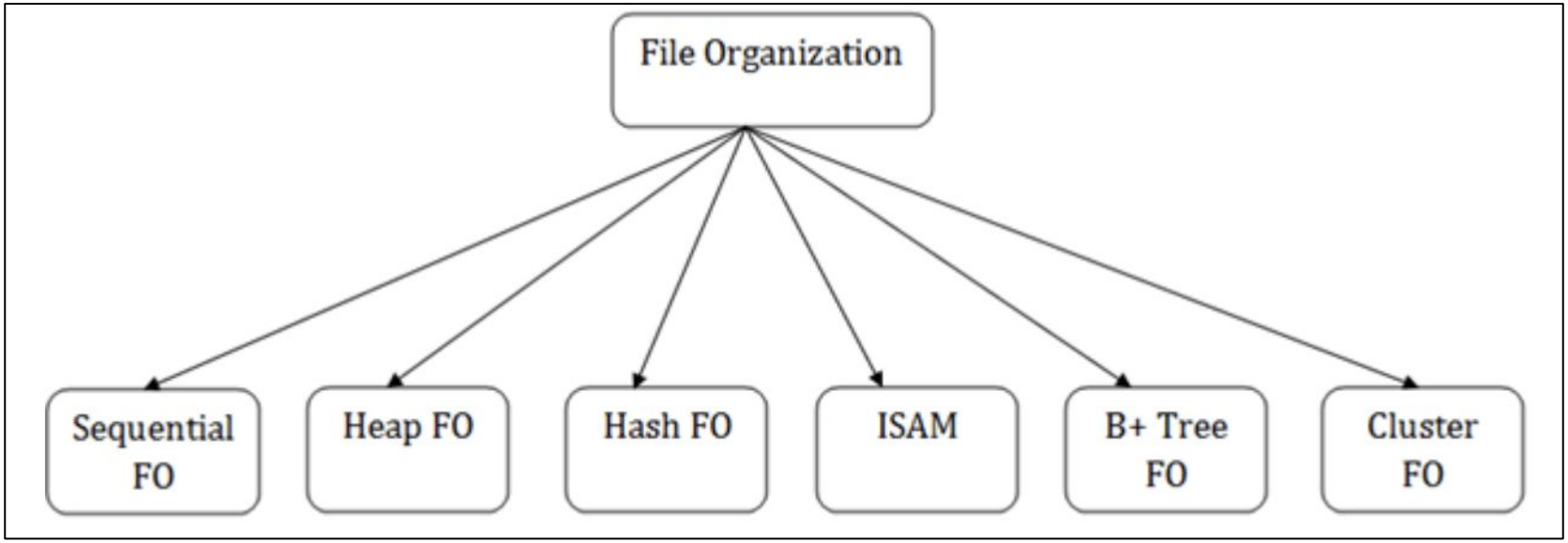
Closing a File

- The concept of closing a file during file handling in c++ refers to the process of detaching a stream with the associated file on disk.
- The file must be closed after performing the required operations on it. Here are some reasons why it is necessary to close the file,
  - The data might be in the buffer after the write operation, so closing a file will cause the data to be written in the file immediately.
  - When you need to use the same stream with another file, it is a good practice to close the previous file.
  - To free the resources held by the file.
- When the object passes out of scope or is deleted, the stream destructor closes the file implicitly.

File Organization

- Files contain records which are collection of information arranged in a specific manner.
- **File organization mainly refers to the logical arrangement of data in a file system**.
- In order to be able to retrieve a target record from a file, it is preferred to be arranged in some defined or proper way.
- It is necessary to organize data records in a particular pattern.
- **The proper arrangement of records within a file is known as file organization.**
- There are various ways in which records in a file can be stored. Files are presented to the application as a stream of bytes and at the end, it contains an EOF (end of file) mark.
- An attribute or combination of attribute values that are used to uniquely identify records within a file is called as a key.
- Keys are used to arrange and/or to retrieve records to/from a file.
- Primary key is one of the keys that can be used to identify a unique record in a file.
- Non-primary keys are called as secondary keys.

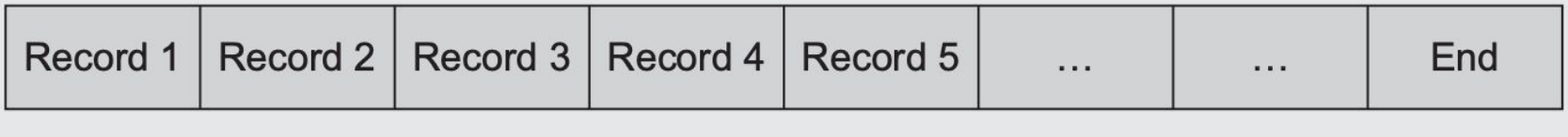
Types of file organization:



- **File access** is a crucial aspect that we can consider while dealing with data.
- It refers to the methods and techniques employed to read and write data from files.
- There are few main types of file access methods: **sequential, direct, and indexed .**
- Each method has its advantages and disadvantages.

Sequential file organization- concept and primitive operations

- A sequential file stores records in the order they are entered.
- The order of the records is fixed.
- The records are stored and sorted in physical, contiguous blocks.
- Within each block, the records are in sequence.
- New records always appear at the end of the file.
- Records in these files can only be read or written sequentially.
- Records may be either fixed or variable in length for this file type.
- This is a significant advantage of sequential files.
- The search time associated with sequential files is more because records are accessed sequentially from the beginning of the file.
- Sequential files are compatible to the magnetic tape storage as shown in the following figure.



Primitive Operations

The following are the primitive operations of the sequential file organization:

1. **Open -** This operation opens the file and sets the file pointer to the first record.
2. **Read - next -** This operation returns the next record to the user. If no record is present, then EOF condition will be set.
3. **Close -** This operation closes the file and terminates access to the file.
4. **Write-next -** File pointers are set to next of last record and this record is written to the file.
5. **EOF -** If EOF condition occurs, this operation returns true, otherwise it returns false.
6. **Search -** This operation searches for the record with a given key.
7. **Update -** The current record is written at the same position with updated values.

The basic file operations : Add

- Adding a record to the sequential file is a one-operation algorithm.
- The new record is simply appended to the end of the file.
- One physical write is required for appending a record in a file.
- Also many records can be collected in the buffer and a block of records can be written at a time in the file.
- The following are the steps involved in addition.
    1. Add a record.
    2. Open a file in append mode.
    3. Read a record from user.
    4. Write a record to the file.
    5. Close the file.

The basic file operations : Search

- A particular record is searched through the file using a key sequentially by comparing with each record key.
- The search starts from the first record and continues till the EOF.
- The following steps are involved in searching:

    1. Open a file in read mode.
    2. Read the value of the record key of the record to be searched.
    3. Read the next record from the file.
    4. If record key = value, display record and go to 7.
    5. If not EOF, then go to 3.
    6. Display ‘Record not found’.
    7. Close the file.

The basic file operations : Delete

- There is no reasonable way to delete records from sequential file. Deletion is done in two ways:
    1. **Logical deletion**
    2. **Physical deletion**
- **Logical deletion -** When disk files are used, records may be **logically deleted by just flagging them as having been deleted.** This can be done by assigning a specific value to one of the attributes of the record. This method needs one extra field to be maintained with each record. The algorithm also needs to modify and check the flag field during operations.
- Another method keeps a record of active and deleted records in a **bit map file**. A bit map is a one-dimensional array in which each bit represents a record in a file. The first bit refers to the first record, and so on. **Bit value '1' tells that the record is active and '0' indicates that the record is deleted.**

The basic file operations : Delete (Logical deletion)

- The following steps are involved in logical deletion:
    1. Open a file in read + write mode.
    2. Read the record key of the record to be deleted.
    3. Read the next record from the file.
    4. If record key = value
        (a) Change status or deleted flag as 1
        (b) Write record back to the same position
        (c) Go to step 7
    5. If not EOF, then go to 3.
    6. Display ‘Record not found’.
    7. Close the file.

The basic file operations : Delete (Physical deletion)

Physical deletion -

- For physical deletion of records, we need to copy the records to another file, skipping the deleted records, and rename the file.
- When the number of the logically deleted records is high, then it is advisable to delete them physically which is known as reorganization of file.
- The following steps are involved in physical deletion (pack):

1. Open a file in read mode.
2. Open ‘temporary’ file in write mode.
3. Read the record key of the record to be deleted.
4. Read the next record from the file.
5. If record key ! = value, write the record to temporary file.
6. If not EOF, then go to 4.
7. Close both the files.
8. Delete the original file.
9. Rename temporary file as original file.
10. Close the file.

The basic file operations : Updation (Modification)

- A record is updated when one or more fields is changed by modifying the information.
- The following steps are involved in updation:

    1. Open a file in write mode.
    2. Read the record key of the record to be modified.
    3. Read the new attributes of the record to be modified.
    4. If record key = value, modify record and go to 7.
    5. If not EOF, then go to 4.
    6. Display ‘Record not found’.
    7. Close the file.

Sequential file organization- Advantages

The following are the main advantages of sequential file organization:

1. Owing to its simplicity, it can be used with a variety of media, including magnetic tapes and disks.
2. It is compatible with variable length records, while most other file organizations are not.
3. Security is ensured with ease.
4. For a run in which a high proportion of a block is hit, as compared to other file organizations, sequential file is efficient specially when processed in batches.

Sequential file organization- Disadvantages

The following are some drawbacks of using sequential file organization:

1. Insertion and deletion of records in in-between positions cause huge data movement.
2. Accessing any record requires a pass through all the preceding records, which is time consuming. Therefore, searching a record also takes more time.
3. Needs reorganization of the file from time to time. If too many records are deleted logically, then the file must be reorganized to free the space occupied by unwanted records.

Direct Access File Organization

- Files that have been designed to make direct record retrieval as easy and efficient as possible are known as directly organized files.
- This is achieved by retrieving a record with a key by getting the address of a record using the key.
- To achieve this, a suitable algorithm, called as hashing, is used to convert the keys to addresses.
- Direct access files are of great use for immediate access to large amounts of information. They are often used in accessing large databases.
- To achieve direct access by having a file size as total number of records, hasing technique is used.
- Hash function generates a natural address from primary key of larger range. for example, MOD (primary key MOD N).
- A synonym is defined as a key, which generates the same address as that generated by a different key.

Direct Access File Organization - Primitive operations

The primitive operations for the dir ect access fi le are as follows:

- **Open -** It opens the fi le and sets the fi le pointer to the fi rst record.
- **Read-next -** It returns the next record to user. If no records are present, then EOF (end of file) condition will be set.
- **Read-direct -** It sets the fi le pointer to a specifi c position and gets the record for the user. If the slot is empty or out of range, then it gives error.
- **Write-direct -** It sets the fi le pointer to a specifi c position and writes the record to file at that position. If the slot is out of range, then it gives error.
- **Update -** Current record is written at the same position with updated values.
- **Close -** This will terminate the access to the fi le.
- **EOF -** If EOF condition occurs, it returns true otherwise it returns false.

- We can use the **fseek()** function for direct access.
- The prototype of **fseek()** is :

int fseek(File *fp , long num-bytes, int origin);

- The **fseek()** function sets the file position indicator. Here **fp** is a file pointer. The **num-bytes** parameter specifies the number of bytes from the **origin** that will become the new current position and **origin** can be one of the following as shown in Table.
- The **fseek()** function returns 0 when successful, and a non-zero value in case of an error.

Origin	Value	Macro name
Beginning of file	0	seek _set
Current position	1	seek_cur
End of file	2	seek _ end

Indexed sequential file organization - concept

- A file that is loaded in key sequence but can be accessed directly by use of one or more indices is known as an indexed sequential file.
- A sequential data file that is indexed is called as indexed sequential file.
- An indexed file contains records ordered by a record key.
- Each record contains a field that contains the record key.
- The record key uniquely identifies the record and determines the sequence in which it is accessed with respect to the other records.
- An indexed file can also use alternate indices, that is, record keys that let you access the file using a different logical arrangement of the records.
- For example, you could access the file through the employee department rather than through the employee number.
- When indexed files are read or written sequentially, the sequence followed is that of the key values.
- Index is a data structure that allows particular records in a file to be located more quickly. An index can be sparse (record for only some of the search key values) or dense (index is maintained for each record), e.g., index in a book

Indexed sequential file organization - concept

Types of Indices

Indices may be of the following three types:

1. **Primary index - It is an index ordered in the same way as the data file**, which is sequentially ordered according to a key. The indexing field is equal to this key.
2. **Secondary index - This is an index that is defined on a non-ordering field of the data file.** In this case, the indexing field need not contain unique values.
3. **Clustering index - A data file can associate with utmost one primary index and several secondary indices.** In this organization, key searches are improved

Structure of Indexed Sequential File

- The file structure is selected according to the physical storage device.
- The external storage device should have the capability to access directly a record as per the key. Devices like magnetic tape can access all records sequentially. The magnetic drum or disk supports direct access.
- **In primary area, actual data records are stored.** Data records are stored as sequential file.
- **The second area is an index area in which the index is stored and is automatically generated.** An index file consists of three areas:
- **Primary storage area -**This includes some unused space to allow for additions made in data.
- **Separate index or indices** - Each query will reference this index first; it will redirect query to part of data file in which the target record is saved.
- **Overflow area -**This is optional separate overflow area.

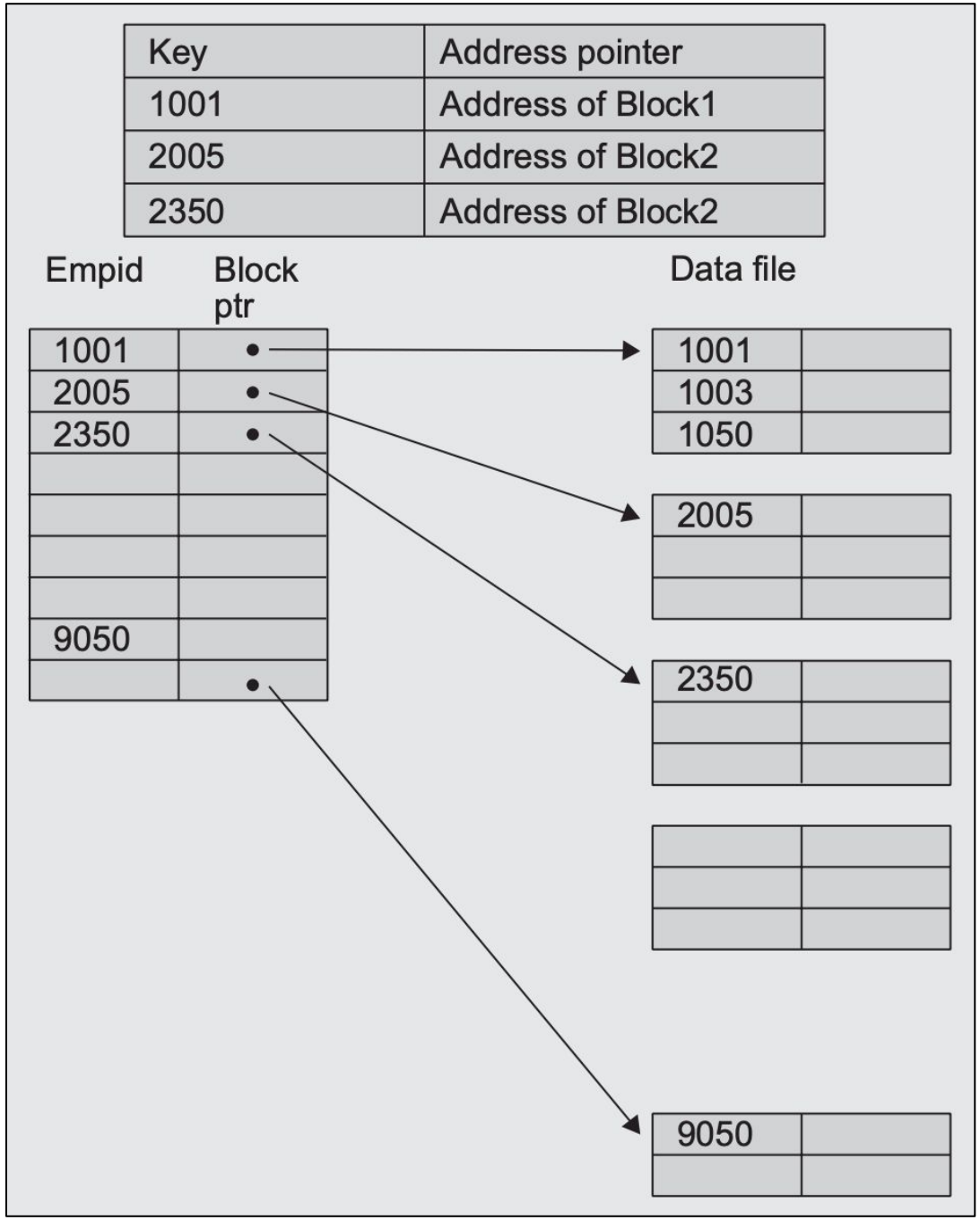
Characteristics of Indexed Sequential File

The following are the characterictics of an indexed sequential file:

1. Records are stored sequentially and a separate index file is maintained for accessing the record directly.
2. Records can be accessed randomly in constant time.
3. Magnetic tape is not suitable for indexed sequential storage.
4. Index is the address of physical storage of a record.
5. When very few records are to be accessed, then indexed sequential file is better.
6. This is a faster access method.
7. Additional overhead is that the index is to be maintained.
8. Indexed sequential files are popularly used in many applications such as a digital library.

Indexed sequential file organization - Example

Consider that an employee file is stored as an indexed sequential file. The entries are as shown in Fig



Indexed sequential file organization

Advantages

1. Accessing any record is more efficient than sequential file organization.

2. Large amount of data can be stored using this type of file organization.

Disadvantage

1. Often more than one index is needed which occupies a large storage area.

Linked Organization

- In linked organization, the physical sequence of records is different from the logical sequence of records.
- The next logical record is obtained by following a link value from the present record.
- Records are linked according to increasing primary key, so insertion and deletion is easy.
- If index is not maintained, then direct searching is difficult and only sequential search is possible.

Linked Organization - Multilist Files

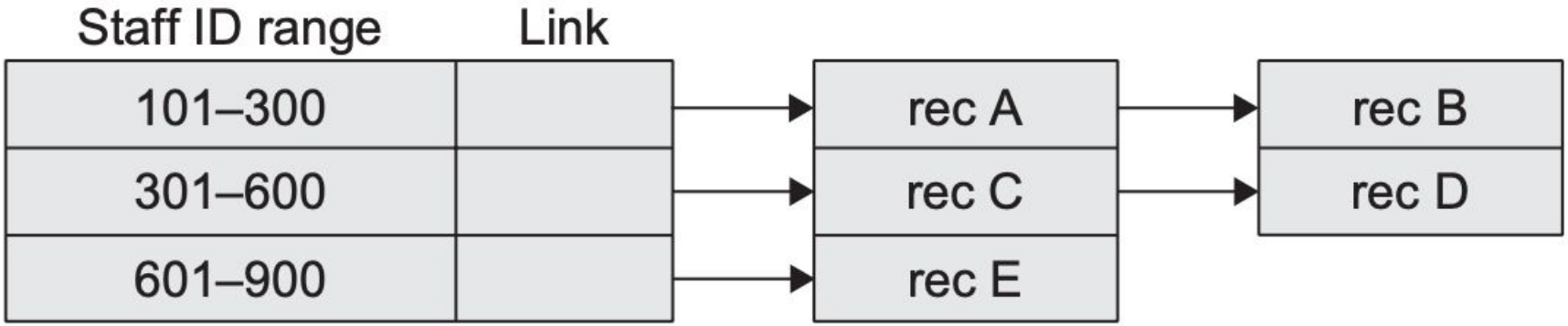
- To make searching easy, several indexes are maintained as per primary key and secondary keys, one index per key.
- The record may be present in different lists as per key.
- Consider the following file of office staff in Table

Staff ID	Occupation	Salary	Record
106	Clerk	5000	A
150	Accountant	4000	B
360	Clerk	3000	C
400	Accountant	3500	D
700	Clerk	2000	E

Linked Organization - Multilist Files

Staff ID	Occupation	Salary	Record
106	Clerk	5000	A
150	Accountant	4000	B
360	Clerk	3000	C
400	Accountant	3500	D
700	Clerk	2000	E

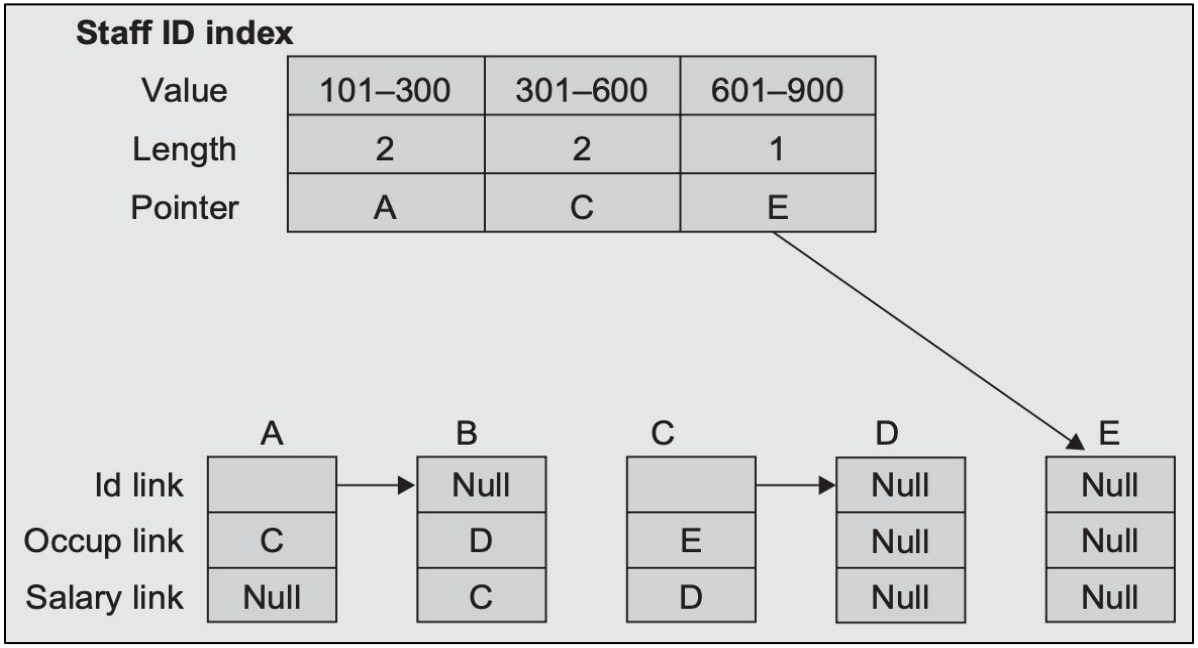
We can maintain indices on the staff ID. We can group staff ID with ranges 101–300, 301–600, 601–900, and so on. Now all the records with staff ID in the same range will be linked together as shown in Fig



Linked Organization - Multilist Files

- We can have multilist structure for file representation by maintaining different indices on different keys and allow records to be in more than one list.
- Suppose indices are maintained on occupation and salary fields, then the multilist structure will look as shown in Fig

Staff ID	Occupation	Salary	Record
106	Clerk	5000	A
150	Accountant	4000	B
360	Clerk	3000	C
400	Accountant	3500	D
700	Clerk	2000	E



Linked Organization - Multilist Files

Table lists the staff details and links for the following values of occupation and salary:

Occupation index

Value	Clerk	Accountant
Length	3	2
Pointer	A	B

Salary index

Value	<= 2000	<= 4000	<= 6000
Length	1	3	1
Pointer	E	B	A

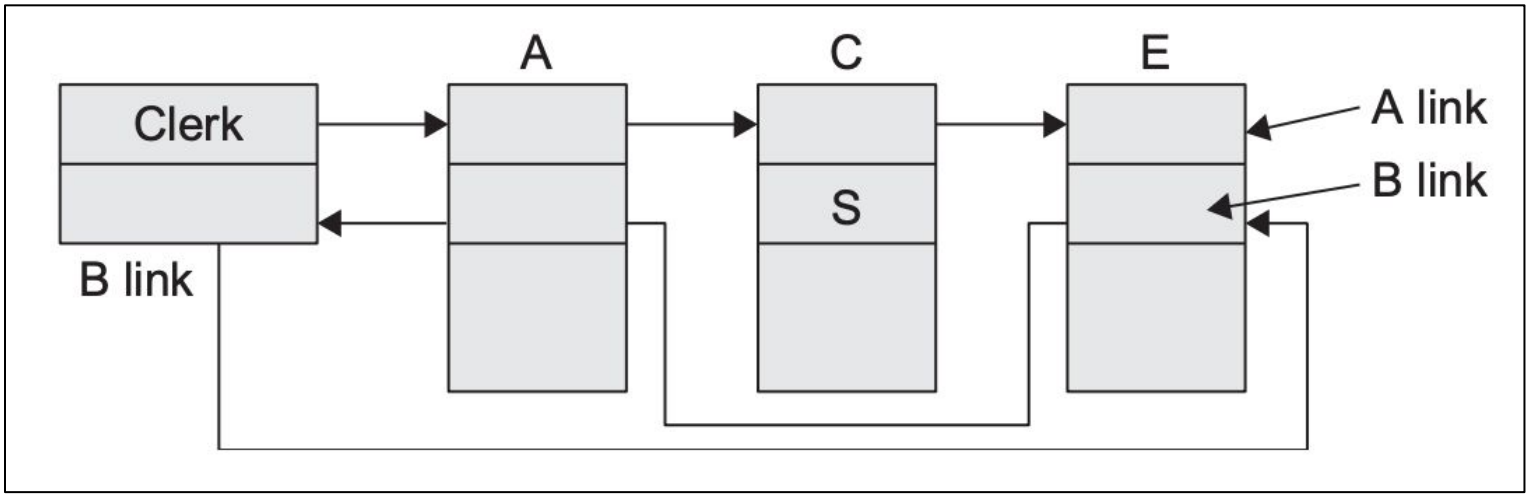
Table of Staff data and links

Record	Staff ID	Occupation	Salary	Occupation link	Salary link
A	106	Clerk	5000	C	–
B	150	Accountant	4000	D	C
C	360	Clerk	3000	E	D
D	400	Accountant	3500	–	–
E	700	Clerk	2000	–	–

- When multilists are maintained, then length of the link is also maintained in the index. When two lists are searched simultaneously, then the search time can be reduced by searching the smaller list.Multilists require storage. So, if space is of importance, then the alternative is the coral ring structure.

Linked Organization - Coral Rings

- In this, doubly linked multilist structure is used as shown in Figure. Each list is circular list with headnode.



- **'A link'** field is used to link all records with same key value.
- '**B link'** is used for some records back pointer and for others it is pointer to head node.
- '**S'** is headnode of the list of 'Clerk'. Owing to these back pointers, deletion is easy without going to start.
- Indexes are maintained as per multilists.

Linked Organization - Inverted Files

- The concept of the inverted files and multilists is similar.
- The difference is that, in multilists records with the same key value are linked together and links are kept in each record.
- But **in the inverted files, the link information is kept in the index itself.**
- For example, consider the same file of office staff used in the link organization. The indices for fully inverted file are shown in Figure

Staff ID index (increasing order)	Occupation index
106	A
150	B
360	C
400	D
700	E

Accountant	B, D
Clerk	A, C, E

Salary index	
2000	E
4000	B, C, D
6000	A

Linked Organization - Inverted Files

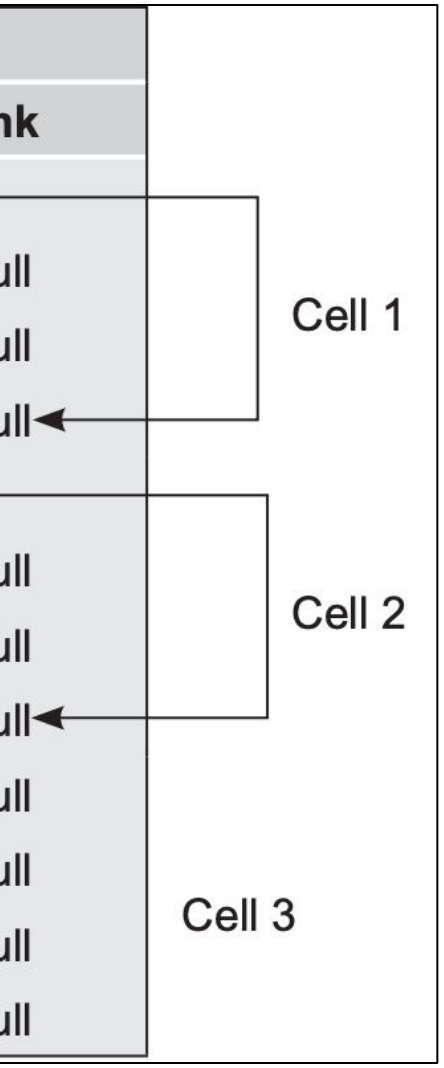
- In inverted files, the record is accessed in two steps.
- First, the indices are searched to obtain a list of required records and then
- Second, records are retrieved using these lists.
- The number of disk accesses required is equal to the number of records being retrieved plus the number to process the indices.
- In inverted files, only the index structures are important.
- The records can be organized sequentially, random, or linked according to primary key.
- Inverted files may also result in space saving when record retrieval does not require retrieval of key fields.
- One of the major disadvantages of the inverted files is that the item values being inverted generally have to be included in both the inverted list and the master file

Linked Organization - cellular partitions

- To decrease file search time, the storage media may be divided into cells.
- A cell may be an entire disk or a cylinder.
- Lists are localized to lie within a cell.
- If a cylinder is used as a cell, then all records on the same cylinder may be accessed without moving the read/write heads.
- We divide multilists organized on several different cylinders into several small lists which are stored on the same cylinder.
- A multilist structure with cellular partitioning is primarily useful when there are a large number of records residing in a cell.
- For cellular multilist structures, index entries may have to be updated with the addition or deletion of records or individual secondary index items.

For example, consider Table , an example of a multilist structure with cellular partitioning for student–teacher data.

	Primary key	Secondary key	
Position	Student ID	Course teacher ID	Link
1	100	A	o
2	200	B	Null
3	300	C	Null
4	400	A	Null
1	500	D	O
2	600	B	Null
3	700	A	Null
4	800	D	Null
1	900	E	Null
2	1000	C	Null
3	1100	D	Null
4	1200	A	Null



- An entry is created in the secondary index whenever the item value occurs one or more times in a cellular partition. The relative secondary index records for the data are shown in Table .

Course teacher ID	Position	Cell no.	Length of link
A	1	1	2
	3	2	1
	4	3	1
B	2	1	1
	2	2	1
C	3	1	1
	2	3	1
D	1	2	2
	3	3	1
E	1	3	1

- The course teacher ID ‘A’ has entries in each cell, at two positions in cell 1, at one position in cell 2, and at one position in cell 3.
- Therefore, the entry of ‘A’ has three rows in the secondary index.
- The course teacher ID ‘E’ has entry only in cell 3 at position 1, so in the secondary index, ‘E’ has only one row.

Course teacher ID	Cell no.
A	1
	2
	3
B	1
	2
C	1
	3
D	2
	3
E	3

Cellular serial structure

Secondary index item	Cell no.		
A	1	1	1
B	1	1	0
C	1	0	1
D	0	1	1
E	0	0	1

Cellular inverted list

Linked Organization - cellular partitions

- A major advantage of cellular partitioning is that several read operations can be initiated simultaneously and these operations can be overlapped with the query processing.
- But the disadvantage is that if there are many records per cell, then access time may be large.