Module – 1 Part B

Fundamental File Structure Concepts, Managing Files of Records

The different types of records are:

- Fixed Length Records.
- Variable Length Records.

Fixed length record: The length of all the records in the file is fixed i.e the length is predefined and altogether the length of the record will not exceed the predefined length.

Variable length record: The length of records is not fixed. Each record are of different length in the file.

2.1 Field & Record Organization

When we are building a file structures, we are making it possible to make data persistent. That is, one program can store data from memory to a file, and terminate. Later, another program can retrieve the data from the file, and process it in memory. In this chapter, we look at file structures which can be used to organize the data within the file, and at the algorithms which can be used to store and retrieve the data sequentially.

Record:

A subdivision of a file, containing data related to a single entity.

Field:

A subdivision of a record containing a single attribute of the entity which the record describes.

Stream of bytes:

A file which is regarded as being without structure beyond separation into a sequential set of bytes.

Field Structures:

There are many ways of adding structure to files to maintain the identity of fields. Four of the most common methods are:

- Fix the length of fields.
- Begin each field with a length indicator.
- Separate the fields with delimiters.
- Use a "Keyword = Value" Expression to identify fields.

Method 1: Fix the length of fields.

In this method length of each field in a record is fixed. For example consider the figure below:

10 bytes	8 bytes	5 Bytes	12 bytes	15 Bytes	20 Bytes
Ames	Mary	123	Maple	Stillwater	OK74075
Mason	Alan	90	Eastgate	Ada	OK74820

Here we have fixed the field 1 size as 10 bytes, field 2 as 8 bytes, field 3 as 5 bytes and so on which results in total length of record to be 70 bytes. While reading the record the first 10 bytes that is read is treated as field 1 the next 8 bytes are treated as field 2 and so on. Disadvantage of this method is padding of each and every field to bring it to pre-defined length which makes the file much larger. Rather than using 4 bytes to store "Ames" we are using 10 bytes. We can also encounter problems with data that is too long to fit into the allocated amount of space.

Method 2: Begin each field with a length indicator.

- he fields within a record are prefixed by a length byte or bytes.
- Fields within a record can have different sizes.
- Different records can have different length fields.
- Programs which access the record must know the size and format of the length prefix.
- There is external overhead for field separation equal to the size of the length prefix per field.

```
04Ames04 Mary0312305Maple10Stillwater07OK74075
05Mason04Alan02 9008Eastgate03Ada07OK74820
```

Method 3: Separate the fields with delimiters.

- The fields within a record are followed by a delimiting byte or series of bytes.
- Fields within a record can have different sizes.
- Different records can have different length fields.
- Programs which access the fields must know the delimiter.
- The delimiter cannot occur within the data.
- If used with delimited records, the field delimiter must be different from the record delimiter.
- Here the external overhead for field separation is equal to the size of the delimiter per field
- Here in the example we have used "|" as delimiter.

```
Ames|Mary|23|Maple|Stillwater|OK74075
Mason|Alan|90|Eastgate|Ada|OK74820
```

Method 4: Use a "Keyword = Value" Expression to identify fields.

- It is the structure in which a field provides information about itself (Self describing).
- It is easy to tell which fields are contained in a file.
- It is a lso good format for dealing with missing fields. If a field is missing, this format makes it obvious, because the keyword is simply not there.
- This format wastes lot of space for storing keywords.

```
Lname=Ames | Fname=Mary | Address=123 Maple | City= Stillwater  | State = OK | Zip=74075
Lname=Mason | Fname=Alan | Address=90 Eastgate | City = Ada | State = OK | Zip = 74820
```

Record Structures:

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

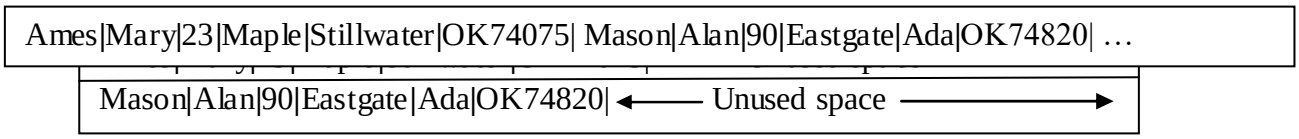
Following are some of the most often used methods for organizing the records of a file.

- Make records a predictable number of bytes.
- Make records a predictable number of fields.
- Begin each record with a length indicator.
- Use an index to keep track of the address.
- Place a delimiter at the end of each record.

Method 1: Make records a predictable number of bytes.

- All records within a file have the same size.
- Programs which access the file must know the record length.
- Offset, or position, of the nth record of a file can be calculated.
- There is no external overhead for record separation.
- There may be internal fragmentation (unused space within records.)
- There will be no external fragmentation (unused space outside of records) except for deleted records.
- Individual records can always be updated in place.
- There are three ways of achieving this method. They are as shown below.

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

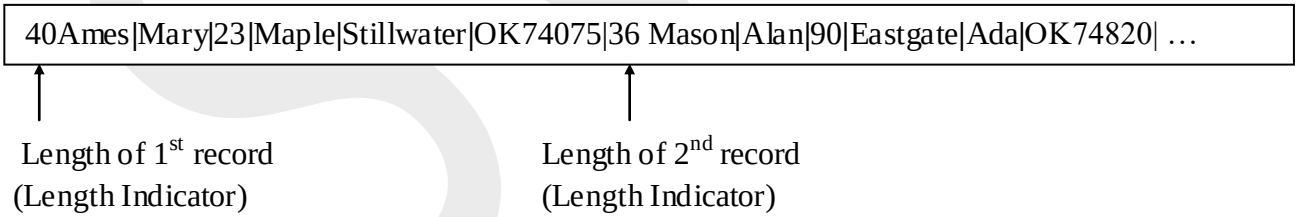


Method 2: Make record a predictable number of fields.

- All the records in the a file contains fixed number of fields.
- In the figure below each record is of 6 fields.

Method 3: Begin each record with a length indicator

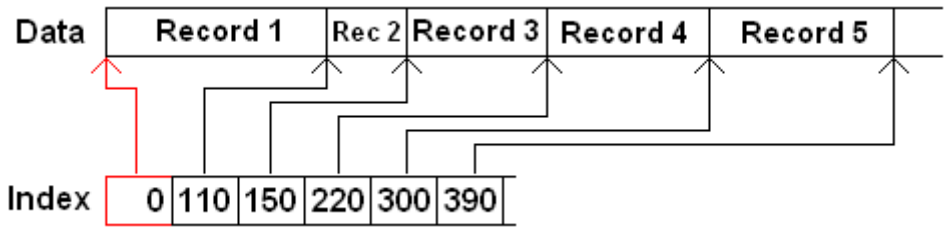
- The records within a file are prefixed by a length byte or bytes.
- Records within a file can have different sizes.
- Different files can have different length records.
- Programs which access the file must know the size and format of the length prefix.
- Offset, or position, of the nth record of a file cannot be calculated.
- There is external overhead for record separation equal to the size of the length prefix per record.



Method 4: Use an index to keep track of the address.

- An auxiliary file can be used to point to the beginning of each record.
- In this case, the data records can be contiguous.
- Disadvantage: there is space overhead for the index file.
- Disadvantage: there is time overhead for the index file.
- The time overhead for accessing the index file can be minimized by reading the entire index file into memory when the files are opened.
- Advantage: there will probably be no internal fragmentation.

- Advantage: the offset of each record is be contained in the index, and can be looked up from its record number. This makes direct access possible.



Method 5: Place a delimiter at the end of each record.

- Records are terminated by a special character or sequence of characters called delimiter.
- Programs which access the record must know the delimiter.
- The delimiter cannot be part of data.
- If used with delimited fields, the field delimiter must be different from the record delimiter.
- Here the external overhead for field separation is equal to the size of the delimiter per record.
- In the following figure we use "|" as field delimiter and "#" as record delimiter.

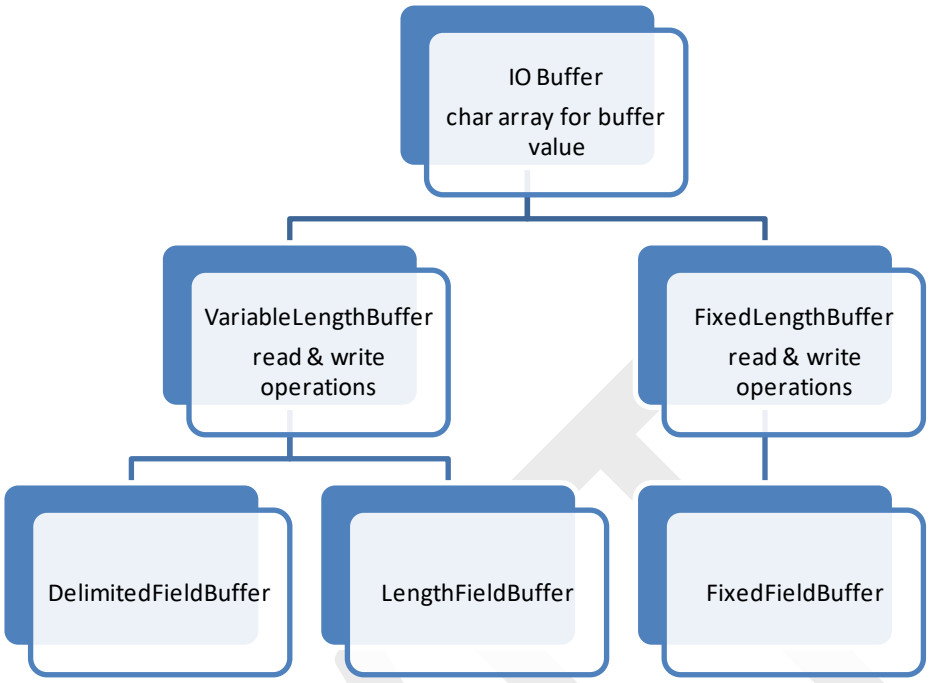
Ames|Mary|23|Maple|Stillwater|OK74075| #Mason|Alan|90|Eastgate|Ada|OK74820|# …

2.2 Using Inheritance for record buffer classes

One or more base classes define members and methods, which are then used by subclasses. Our discussion in this section deals with fstream class which is embedded in a class hierarchy that contains many other classes. The read operations, including the extraction operators are defined in class istream. The write operations are defined in class ostream. Class iostream, inherits from istream and ostream.

A Class Hierarchy for Record Buffer Objects

The members and methods that are common to all of the three buffer classes are included in base class IOBuffer. Example the members in IOBuffer class like BufferSize, MaxBytes, NextByte are inherited in its derived class VariableLengthBuffer and FixedLengthBuffer. Similarly methods read(), Write(), Pack(), Unpak() of IOBuffer are inherited by derived classes VariableLengthBuffer and FixedLengthBuffer. (Refer Text book for class definition).



Here the member functions read(), Write(), Pack(), Unpak() of class IOBuffer are virtual functions so that subclasses VariableLengthBuffer and FixedLengthBuffer define its own implementation. This means that the class IOBuffer does not include an implementation of the method. The reading and writing of variable length records are included in the class VariableLengthBuffer. Packing and Unpacking delimited fields is defined in the class DelimitedFieldBuffer which is derived from VariableLengthBuffer.

2.3 Record Access

When looking for an individual record, it is convenient to identify the record with a key based on record contents. The key should be unique so that duplicate entries can be avoided. Foe example, in the previous section example we might want to access the "Ames record" or the "Mason record" rather than thinking in terms of the "first record" or "second record".

When we are looking for a record containing the last name Ames we want to recognize it even if the user enters the key in the form "AMES", "ames" or "Ames". To do this we must define a standard form of keys along with associated rules and procedures for converting keys into this standard form. This is called as canonical form of the key.

Sequential Search

Reading through the file, record by record, looking for a record with a particular key is called sequential searching. In general the work required to search sequentially for a record in a file with 'n' records in proportional to n: i.e the sequential search is said to be of the order O(n).

This efficiency is tolerable if the searching is done on the date present in the main memory, but not for, which has to be extracted from secondary storage device, due to high delay involved in accessing the data. Instead of extracting the records from the secondary storage device one at a time sequential we can access some set of records at once from the hard disk, store it in main memory and do the comparisons. This is called as record blocking.

In some cases sequential search is superior like:

Repetitive hits: Searching for patterns in ASCII files.

Searching records with a certain secondary key value.

Small Search Set: Processing files with few records.

Devices/media most hospitable to sequential access: tape, binary file on disk.

Unix Tools For Sequential Processing

Some of the UNIX commands which perform sequential access are:

Cat: displays the content of the file sequentially on the console.

%cat filename

Example: %cat myfile

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

wc: counts the number of words lines and characters in the file

%wc filename

Example: %wc myfile

2 14 76

grep: (generalized regular expression) Used for pattern matching

%grep string filename

Example: % grep Ada myfile

Mason	Alan	90	Eastgate	Ada	OK74820

Direct Access:

The most radical alternative to searching sequentially through a file for a record is a retrieval mechanism known as direct access. The major problem with direct access is knowing where the beginning of the required record is. One way to know the beginning of the required record or byte

offset of the required record is maintaining a separate index file. The other way is by relative record number RRN. If a file is a sequence of records, the RRN of a record gives its position relative to the beginning of the file. The first record in a file has RRN 0, the next has RRN 1, and so forth.

To support direct access by RRN, we need to work with records of fixed length. If the records are all the same length, we can use a record's RRN to calculate the byte offset of the start of the record relative to the start of the file. For example if we are interested in the record with an RRN of 546 and our file has fixed length record size of 128 bytes per record, we can calculate the byte offset of a record with an RRN of n is

Byte offset = 546 * 128 = 69888

In general Byte offset = n * r where r is length of record.

Header Records

It is often necessary or useful to keep track of some general information about a file to assist in future use of the file. A header record is often placed at the beginning of the file to hold information such as number of records, type of records the file contains, size of the file, date and time of file creation, modification etc. Header records make a file self describing object, freeing the software that accesses the file from having to know a priori everything about its structure. The header record usually has a different structure than the data record.

2.4 File Access and File Organization

- File organization is static.
    - Fixed Length Records.
    - Variable Length Records.
- File access is dynamic.
    - Sequential Access
    - Direct Access.