FIELD AND RECORD ORGANIZATION

4.1 Field and Record Organization

4.1.1 A Stream File

- In the Windows, DOS, UNIX, and LINUX operating systems, files are not internally structured; they are streams of individual bytes.

F	r	e	d		F	l	i	n	t	s	t	o	n	e	4	4	4	4		G	r	a	n	...

- The only file structure recognized by these operating systems is the separation of a text file into lines.
    - For Windows and DOS, two characters are used between lines, a carriage return (ASCII 13) and a line feed (ASCII 10);
    - For UNIX and LINUX, one character is used between lines, a line feed (ASCII 10);
- The code in applications programs can, however, impose internal organization on stream files.

Record Structures

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.

4.2 Using Classes to Manipulate Buffers

- Within a program, data is temporarily stored in variables.
- Individual values can be aggregated into structures, which can be treated as a single variable with parts.
- In C++, classes are typically used as as an aggregate structure.
- C++ Person class (version 0.1):

```
class Person {
  public:
    char FirstName [11];
    char LastName[11];
```

```
    char Address [21];
    char City [21];
    char State [3];
    char ZIP [5];
};
```

- With this class declaration, variables can be declared to be of type Person. The individual fields within a Person can be referred to as the name of the variable and the name of the field, separated by a period (.).
- C++ Program:

```
#include
class Person {
  public:
    char FirstName [11];
    char LastName[11];
    char Address [31];
    char City [21];
    char State [3];
    char ZIP [5];
};

void Display (Person);
int main () {
  Person Clerk;
  Person Customer;
  strcpy (Clerk.FirstName, "Fred");
  strcpy (Clerk.LastName, "Flintstone");
  strcpy (Clerk.Address, "4444 Granite Place");
  strcpy (Clerk.City, "Rockville");
  strcpy (Clerk.State, "MD");
  strcpy (Clerk.ZIP, "00001");
  strcpy (Customer.FirstName, "Lily");
  strcpy (Customer.LastName, "Munster");
  strcpy (Customer.Address, "1313 Mockingbird Lane");
  strcpy (Customer.City, "Hollywood");
  strcpy (Customer.State, "CA");
  strcpy (Customer.ZIP, "90210");
```

```
  Display (Clerk);
  Display (Customer);
}
void Display (Person Someone) {
  cout << Someone.FirstName << Someone.LastName
       << Someone.Address << Someone.City
       << Someone.State << Someone.ZIP;
}
```

- In memory, each Person will appear as an aggregate, with the individual values being parts of the aggregate:

Person					
Clerk					
FirstName	**LastName**	**Address**	**City**	**State**	**ZIP**
Fred	Flintstone	4444 Granite Place	Rockville	MD	0001

- The output of this program will be:

  FredFlintstone4444 Granite PlaceRockvilleMD00001LilyMunster1313 Mockingbird LaneHollywoodCA90210

- Obviously, this output could be improved. It is marginally readable by people, and it would be difficult to program a computer to read and correctly interpret this output.

Source : http://elearningatria.files.wordpress.com/2013/10/ise-vi-file-structures-10is63-notes.pdf