Chapter 4 Fundamental File Structure Concepts

Fields and Records

Classes for buffers of records

A stream: << (in C++)

Just put the characters to the file designated.

There is no way to get it apart once the data is in the stream

We loose the fundamental pieces

Fields: The smallest meaningful unit of information in a file

So how do we change the way we view data or write it out so we can get it back?

This is the question we need to answer in this class.

There are 4 basic methods for organizing fields so we can retrieve the data.

1. Each field is given a fixed length

```
Ames     Mary     123 Maple       Stillwater     OK74075
Mason    Alan     90 Eastgate     Ada            MN54820
```

- We have to pad (with blanks) when the contents does not need all the space
- We have to truncate when the data is too long for the space.
- This is the easiest technique so it is appropriate to use if the data has a natural size.

2. Each field begins with a length indicator (Length-Based)

```
04Ames04Mary09123 Maple10Stillwater02OK0574075
05Mason04Alan1190 Eastgate03Ada02MN0554820
```

- If we are lucky we can store the length in 1 byte (255)
- It takes time to calculate the lengths
- This works well for data that has a natural wide variation in size

3. Separate the fields with delimiters

```
Ames|Mary|123 Maple|Stillwater|OK|74075|
Mason|Alan|90 Eastgate|Ada|MN|54820
```

- the delimiter cannot be something that would occur in the data
- white-space, blank, newline, tab can often be used in character data
- but can't be used in street addresses
- filter input to fields for delimiter character

4. Use "Keyword = Value" expressions

```
last=Ames|first=Mary|address=123 Maple|city=Stillwater|
state=OK|zip=74075|
last=Mason|first=Alan|address=90 Eastgate|city=Ada|
state=MN|zip54820
```

- Still requires a delimiter but the fields can be written in any order
- Takes a lot of space
- looking at a dump will tell you what is in the file
- programs don't have to know how the data is stored

Records: Collection of fields that we want, sometimes, to treat as an aggregate.

Records can be fixed in length or variable in length.

How do we organize the records so that we know when one record stops and the next begins?

- Fixed Length Records
  - Requires the records be a fixed number of bytes in length
  - Most common because they are the easiest
  - Within each record we can use any of the field storage techniques

```
Ames     Mary     123 Maple       Stillwater     OK74075
Mason    Alan     90 Eastgate     Ada            MN54820
```

Use fixed length fields

```
Ames|Mary|123 Maple|Stillwater|OK|74075|
Mason|Alan|90 Eastgate|Ada|MN|54820|
```

Use delimited fields

```
04Ames04Mary09123 Maple10Stillwater02OK0574075
05Mason04Alan1190 Eastgate03Ada02MN0554820
```

Use field counts

- Variable length record fixed number of fields
  - Use delimiters or counts to distinguish the fields

```
Ames|Mary|123 Maple|Stillwater|OK|74075| Mason|Alan|90 Eastgate
|Ada|MN|54820|
```

```
04Ames04Mary09123 Maple10Stillwater02OK057407505Mason04Alan1190
Eastgate03Ada02MN0554820
```

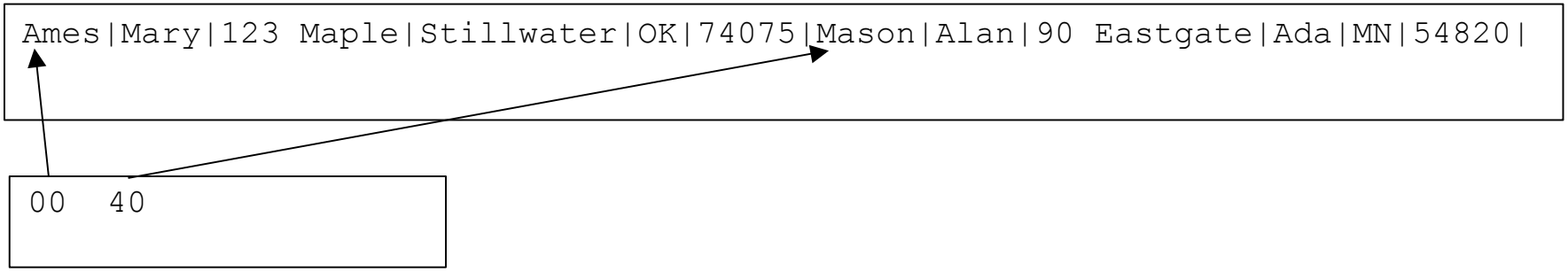
- Each record has a length indicator
- can use counts or delimiters to distinguish the fields
- you don't need a fixed number of fields

```
40Ames|Mary|123 Maple|Stillwater|OK|74075|36Mason|Alan|
90 Eastgate|Ada|MN|54820|
```

- Use an index to keep track of the start of each record

  You can use this to figure out how long the record is

  Fields use delimiters or counts



- End each record with a delimiter

```
Ames|Mary|123 Maple|Stillwater|OK|74075|#Mason|Alan|90 Eastgate
|Ada|MN|54820|#
```