Basics

- some definitions
- basic file organization
- physical files and logical files
- field and stream
- basic file operations
  - assign/open/create
  - close
  - read
  - write
  - eof
- ASCII files and binary files
- access: sequential vs. direct

Definitions

File

1. A collection of data on some secondary storage device
2. A collection of *records*

Record

A collection of *fields*

Field

A collection of characters (bytes)

Key

A subset of the *fields* in a *record* used to identify (uniquely, usually) the record

Basic File Organization

File

Record1	Field1	**Field2**	**Field3**	...	FieldN
Record2	Field1	**Field2**	**Field3**	...	FieldN
Record3	Field1	**Field2**	**Field3**	...	FieldN
.					
.	...	**...**	**...**	...	...
.					
RecordM	Field1	**Field2**	**Field3**	...	FieldN

*Key = {***Field2**, **Field3***}* */* for example */*

For Example

```
{kamla}kbarker(42) cat courses.dat

BOSE   JIT     CARLETONU CS384   DATA STRUCTURES          .
BOSE   JIT     CARLETONU CS102   SYSTEMS PROGRAMMING      .
BARKER KEN     UMANITOBA 074-438 DATABASE IMPLEMENTATION  .
BARKER KEN     UMANITOBA 074-452 PROJECTS                 .
BARKER KEN     UOTTAWA   CSI2131 FILE MANANGEMENT         .
BARKER KEN     UOTTAWA   CSI4900 PROJECTS                 .
BOYD   SYLVIA  UOTTAWA   CSI5166 COMBINATORICS            .
BOYD   SYLVIA  UOTTAWA   CSI4900 PROJECTS                 .
HOLTE  ROBERT  UOTTAWA   CSI1101 COMPUTER SCIENCE II      .
HOLTE  ROBERT  UOTTAWA   CSI4900 PROJECTS                 .
ROY    DAMIEN  UOTTAWA   MAT1741 ALGÈBRE LINÉAIRE         .
ROY    DAMIEN  UOTTAWA   MAT3543 STRUCTURES ALGÉBRIQUES   .
ROY    LANGIS  UOTTAWA   ELG4102 µWAVE & OPTICAL CIRCUITS.
MORIN  JOHANNE UQTR      ASY1006 GÉNIE LOGICIEL           .
MORIN  JOHANNE UQTR      SIF1016 STRUCTURES DE DONNÉES    .
```

Q: *What are the* fields *in* `courses.dat`*?*

A:

Q: *What are the* records *in* `courses.dat`*?*

A:

Q: *What would be a* unique key *for* `courses.dat`*?*

A:

Variable Length Fields

```
{kamla}kbarker(43) cat courses-varfields.dat

BOSE/JIT/CARLETONU/CS384/DATA STRUCTURES                .
BOSE/JIT/CARLETONU/CS102/SYSTEMS PROGRAMMING            .
BARKER/KEN/UMANITOBA/074-438/DATABASE IMPLEMENTATION.
BARKER/KEN/UMANITOBA/074-452/PROJECT                    .
BARKER/KEN/UOTTAWA/CSI2131/FILE MANANGEMENT             .
BARKER/KEN/UOTTAWA/CSI4900/PROJECTS                     .
BOYD/SYLVIA/UOTTAWA/CSI5166/COMBINATORICS               .
BOYD/SYLVIA/UOTTAWA/CSI4900/PROJECTS                    .
HOLTE/ROBERT/UOTTAWA/CSI1101/COMPUTER SCIENCE II        .
HOLTE/ROBERT/UOTTAWA/CSI4900/PROJECTS                   .
ROY/DAMIEN/UOTTAWA/MAT1741/ALGÈBRE LINÉAIRE             .
ROY/DAMIEN/UOTTAWA/MAT3543/STRUCTURES ALGÉBRIQUES       .
ROY/LANGIS/UOTTAWA/ELG4102/µWAVE & OPTICAL CIRCUITS .
MORIN/JOHANNE/UQTR/ASY1006/GÉNIE LOGICIEL               .
MORIN/JOHANNE/UQTR/SIF1016/STRUCTURES DE DONNÉES        .
```

Other Possibilities

Store the length of each field *right in the file*

```
06BARKER03KEN07UOTTAWA07CSI213116FILE MANANGEMENT
05MORIN07JOHANNE04UQTR07ASY100614GÉNIE LOGICIEL
...
```

Store each field as a *value = expression;*

```
ln=BARKER;fn=KEN;un=UOTTAWA;cc=CSI2131;ti=FILE MANANGEMENT;
ln=MORIN;fn=JOHANNE;un=UQTR;cc=ASY1006;ti=GÉNIE LOGICIEL;
...
```

Variable Length Records

```
{kamla}kbarker(44) cat courses-varrec.dat

BOSE/JIT/CARLETONU/CS384/DATA STRUCTURES.BOSE/JIT/CAR
LETONU/CS102/SYSTEMS PROGRAMMING.BARKER/KEN/UMANITOBA
/074-438/DATABASE IMPLEMENTATION.BARKER/KEN/UMANITOBA
/074-452/PROJECT.BARKER/KEN/UOTTAWA/CSI2131/FILE MANA
NGEMENT.BARKER/KEN/UOTTAWA/CSI4900/PROJECTS.BOYD/SYLV
IA/UOTTAWA/CSI5166/COMBINATORICS.BOYD/SYLVIA/UOTTAWA/
CSI4900/PROJECTS.HOLTE/ROBERT/UOTTAWA/CSI1101/COMPUTE
R SCIENCE II.HOLTE/ROBERT/UOTTAWA/CSI4900/PROJECTS.RO
Y/DAMIEN/UOTTAWA/MAT1741/ALGÈBRE LINÉAIRE.ROY/DAMIEN/
UOTTAWA/MAT3543/STRUCTURES ALGÉBRIQUES.ROY/LANGIS/UOT
TAWA/ELG4102/µWAVE & OPTICAL CIRCUITS.MORIN/JOHANNE/U
QTR/ASY1006/GÉNIE LOGICIEL.MORIN/JOHANNE/UQTR/SIF1016
/STRUCTURES DE DONNÉES.
```

Other Possibilities

Store the length of each record *right in the file*

```
043BARKER/KEN/UOTTAWA/CSI2131/FILE MANANGEMENT
041MORIN/JOHANNE/UQTR/ASY1006/GÉNIE LOGICIEL
...
```

Use a separate file with the starting position of each record

```
000;043;084;...

BARKER/KEN/UOTTAWA/CSI2131/FILE MANANGEMENT
MORIN/JOHANNE/UQTR/ASY1006/GÉNIE LOGICIEL
...
```

Variable vs. Fixed

The advantages of variable length fields and records are obvious:

- unlimited field/record length
- more compact storage

The advantages of fixed length fields and records are less obvious.

- simpler, more "readable" representation
- can jump directly to a particular record/field
- often, the *data structures* we use in our programs are fixed in length; a file with fixed length fields and records can be read directly into fixed length data structures (we'll see this later)

Pascal	*C*

`TYPE`
`  course_descrip = RECORD`
`  lname: array [1..7]  of char;`
`  fname: array [1..9]  of char;`
`  univ : array [1..10] of char;`
`  code : array [1..8]  of char;`
`  title: array [1..25] of char;`
`END;`
	`struct`
	`{`
	`  char lname[7];`
	`  char fname[9];`
	`  char univ[10];`
	`  char code[8];`
	`  char title[25];`
	`} course_descrip;`

Physical Files and Logical Files

- A *physical file* is a collection of bytes sitting on a disk or tape, etc.

- A *logical file* is a connection through which a program does input and output.
  - the program can open and close a logical file, read from it, write to it, etc.
  - most programming languages handle *all* I/O (including keyboard input, screen output, printer output, etc.) using logical files.

- One of the most common uses of logical files in a program is to access the data in physical files.

Field and Stream

If we want to treat all input sources and output targets consistently (as logical files), we need a consistent way to access them. The most common way is to treat files as *streams*. The idea of a *stream file* is simple:

- a logical input file is seen as a stream of bytes flowing into the program one-by-one.
  - the program reads the current byte from the stream and then the next byte in the physical file (or input device) becomes the current byte that will be read by the next read statement.

- a logical output file is seen as a stream of bytes flowing out of the program one-by-one.
  - the program writes a byte to the current position in the output file and then the next byte written will be placed after the previous one in the physical file or output device.

Q: *If programs access files one byte at a time, how do they access* fields *in files?*

A:

Q: *Streams sure look elegant and convenient, but what about practical matters of efficiency in the real-life dog-eat-dog world of file management?*

A:

Connecting Logical Files to Physical Files

For this whole logical file/physical file thing to work, we need to tell the system which physical file (or I/O device) a logical file refers to.

Many programming languages combine the physical↔logical assignment and `file open` in one command.

COBOL

```
...
// ASSGN SYS041,DISK,VOL=SAG03P,SHR
...
```

```
IDENTIFICATION DIVISION.
...
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INFILE ASSIGN TO SYS041.
...
```

Pascal

```
...
assign(infile, 'foo.dat');
reset(infile);
...
```

C

```
...
infile = fopen("foo.dat", "r");
...
```

N.B. *You must* open *a file before you can read from it or write to it... and you must* close *it when you're done.*

Simple Open, Read/Write, Close

Pascal

```
var
 ch  : char;
 ifil, ofil: file of char;

begin
 assign(ifil, 'foo.dat');
 reset(ifil);

 assign(ofil, 'bar.dat');
 rewrite(ofil);

 while not (eof(ifil)) do
    begin
     read(ifil, ch);
     write(ofil, ch)
    end;

 close(ifil);
 close(ofil)
end;
```

C

```
{
 char ch;
 FILE *ifil, *ofil;


 ifil = fopen("foo.dat", "r");


 ofil = fopen("bar.dat", "w");

 ch = getc(ifil);
 while(ch != EOF)
    {
     putc(ch, ofil);
     ch = getc(ifil);
    }

 fclose(ifil);
 fclose(ofil);
}
```

Of course, both languages have many flavours of read and write, which you will get to know intimately.

♥

ASCII Files and Binary Files

Consider the file `eruptions.dat`:

```
{kamla}kbarker(45) cat eruptions.dat

Etna              Italy          3315  1996
Fuji              Japan          3776  1707
Kick-'em-Jenny    Grenada        -160  1939
Krakatau          Java/Sumatra    813   416
Momotombo         Nicaragua      1258  1524
Pinatubo          Philippines    1600  1991
St. Helens        USA            2549  1980
Vesuvius          Italy          1281    79
```

The information could be represented in the following data structure:

Pascal

```
TYPE
 eruption = RECORD
  volcano: array [1..15] of char;
  country: array [1..13] of char;
  elev   : array [1..5]  of char;
  date   : array [1..5]  of char;
END;
```

C

```
struct
{
 char volcano[15];
 char country[13];
 char elev[5];
 char date[5];
} eruption;
```

ASCII Files and Binary Files (cont.)

But doesn't this make more sense?

Pascal

```
TYPE
 eruption = RECORD
  volcano: array [1..15] of char;
  country: array [1..13] of char;
  elev   : integer;
  date   : integer;
END;
```

C

```
struct
{
 char volcano[15];
 char country[13];
 int elev;
 int date;
} eruption;
```

This data structure represents elevations and dates as numbers, not strings of character digits. Elevations and dates will be stored as two-byte binary numbers, instead of five bytes of ASCII character codes.

E.g., the elevation of Momotombo will be represented by the bits:

$$0000010011101010_2 = 1258_{10}$$

instead of:

00100000_2 00110001_2 00110010_2 00110101_2 00111000_2

= ' ' = '1' = '2' = '5' = '8'

□

Q: *What are the advantages of this new representation?*

A:

ASCII Files and Binary Files (cont.)

But how would the binary representation look printed directly to the screen?

```
{kamla}kbarker(46) cat eruptions.bin

Etna              Italy          ^LóÌ
Fuji              Japan          ^NÀ^F«
Kick-'em-Jenny    Grenada        ÿ`□
Krakatau          Java/Sumatra   ^C-^A
Momotombo         Nicaragua      ^Dê^Eô
Pinatubo          Philippines    ^F@Ç
St. Helens        USA                 õ¼
Vesuvius          Italy          ^E^A^@O
```

□

Q: *What the?*

ASCII Files and Binary Files (cont.)

A file is a just collection of bytes, each of which is a number between 0 and 255.

ASCII (American Standard Code for Information Interchange) is a code that associates some of those numbers with character symbols corresponding to the letters of the alphabet, digits, punctuation, etc.

The Unix command `cat` (just like the DOS command `type`) attempts to interpret all files as *ASCII* files. That is, `cat` assumes that it is meaningful to convert the bytes in a file to the character symbols using the ASCII number-to-character map.

In displaying the file `eruptions.bin`, `cat` assumed that the four bytes at the end of each record corresponded to four ASCII characters and tried to display those four symbols. Unbeknownst to the simple-minded `cat`, however, those four bytes are actually the binary representation of two integers.

Example

Pascal

```
program message(input, output);

const
   mess   : array [1..8] of byte =
            (174, 222, 222, 208, 222, 222, 66, 20);
var
   i      : integer;
   byteout: byte;
   outfil : file of byte;

begin
   assign(outfil, 'mess.out');
   rewrite(outfil);

   for i := 1 to 8 do begin
      byteout := mess[i] div 2;
      write(outfil, byteout)
   end;

   close(outfil)
end.
```

Example (cont.)

After running the program, we get:

```
{kamla}kbarker(47) cat mess.out
Woohoo!
```

- In general, a file is considered an *ASCII* file, or a *text* file if it is meaningful to interpret the complete file by mapping bytes to characters according to the ASCII map.

- In general, a file is considered a *binary* file if interpreting bytes as ASCII characters is not meaningful.

Q: *Would the following files likely be ASCII files or binary files?*

- `eruptions.dat` *?*
- `mario.exe` *?*
- `eruptions.bin` *?*
- `readme.txt` *?*
- `thesis.doc` *?*

Organization and Access

Organization

Organization refers to the way files are *laid out* in secondary storage (disks, tapes, etc.).

Access

Access refers to the way elements within a file can be reached by software.

- *The existing organization of a file will dictate how it can be accessed.*

- *The access requirements to the information in a file will dictate how it must be organized.*

Sequential Organization

- Records appear in the file in the order in which they are added to the file.
- The physical order of records may correspond to some logical ordering of the records.

Recall the `courses.dat` file:

```
BOSE   JIT     CARLETONU CS384   DATA STRUCTURES          .
BOSE   JIT     CARLETONU CS102   SYSTEMS PROGRAMMING      .
BARKER KEN     UMANITOBA 074-438 DATABASE IMPLEMENTATION  .
BARKER KEN     UMANITOBA 074-452 PROJECT                  .
BARKER KEN     UOTTAWA   CSI2131 FILE MANANGEMENT         .
BARKER KEN     UOTTAWA   CSI4900 PROJECTS                 .
BOYD   SYLVIA  UOTTAWA   CSI5166 COMBINATORICS            .
BOYD   SYLVIA  UOTTAWA   CSI4900 PROJECTS                 .
HOLTE  ROBERT  UOTTAWA   CSI1101 COMPUTER SCIENCE II      .
HOLTE  ROBERT  UOTTAWA   CSI4900 PROJECTS                 .
ROY    DAMIEN  UOTTAWA   MAT1741 ALGÈBRE LINÉAIRE         .
ROY    DAMIEN  UOTTAWA   MAT3543 STRUCTURES ALGÉBRIQUES   .
ROY    LANGIS  UOTTAWA   ELG4102 µWAVE & OPTICAL CIRCUITS.
MORIN  JOHANNE UQTR      ASY1006 GÉNIE LOGICIEL           .
MORIN  JOHANNE UQTR      SIF1016 STRUCTURES DE DONNÉES    .
```

If a new record is added, it is simply stuck on the end of the file.

Sequential Access

- Records are retrieved from the file in the order in which they appear in the file.
- Record r_i is retrieved after i - 1 records have been retrieved (*i.e.*, after retrieving records $r_1, r_2, \ldots, r_{i-1}$)

Q: *What are the advantages of sequential organization?*

A:

Q: *What are the advantages of sequential access?*

A:

Q: *What are the disadvantages of sequential organization?*

A:

Q: *What are the disadvantages of sequential access?*

A:

Relative Organization

- Records appear in positions in the file that can be determined using the record keys.
- The physical order of records may be completely indpendent of any logical ordering of the records.

Suppose course offerings are identified by a "C Number":

```
C01 BOSE   JIT     CARLETONU CS102   SYSTEMS PROG...
C02 BARKER KEN     UMANITOBA 074-438 DATABASE IMP...
C03 BOSE   JIT     CARLETONU CS384   DATA STRUCTU...
C04 BARKER KEN     UOTTAWA   CSI2131 FILE MANANGE...
C05 HOLTE  ROBERT  UOTTAWA   CSI1101 COMPUTER SCI...
C06 BOYD   SYLVIA  UOTTAWA   CSI5166 COMBINATORIC...
C07 ROY    DAMIEN  UOTTAWA   MAT1741 ALGÈBRE LINÉ...
C08 BOYD   SYLVIA  UOTTAWA   CSI4900 PROJECTS    ...
C09 BARKER KEN     UMANITOBA 074-452 PROJECT     ...
C10 HOLTE  ROBERT  UOTTAWA   CSI4900 PROJECTS    ...
C11 MORIN  JOHANNE UQTR      ASY1006 GÉNIE LOGICI...
C12 ROY    DAMIEN  UOTTAWA   MAT3543 STRUCTURES A...
C13 ROY    LANGIS  UOTTAWA   ELG4102 µWAVE & OPTI...
C14 BARKER KEN     UOTTAWA   CSI4900 PROJECTS    ...
C15 MORIN  JOHANNE UQTR      SIF1016 STRUCTURES D...
```

Records are placed such that their "C Number" corresponds to their position in the file.

Relative Access

- Records are retrieved from the file by computing the relative address of the file based on the key.
- Record r_i is retrieved by providing the key k_i of r_i.

If we know the "C Number" of the course offering we're interested in, we can seek directly to the correct position in the file using the formula:

```
offset = (CNumber - 1) × RecordLength
```

□

Q: *What are the advantages of relative organization?*

A:

Q: *What are the advantages of relative access?*

A:

Q: *What are the disadvantages of relative organization?*

A:

Q: *What are the disadvantages of relative access?*

A:

Indexed Sequential Organization

- Records appear in logically consecutive positions in the file based on some key.
- An *index file* relates values of the key to positions in the file.

Index File

```
CARLETONU  0000
UMANITOBA  0116
UOTTAWA    0232
UQTR       0754
```

Data File

```
BOSE   JIT     CARLETONU CS384   DATA ST...
BOSE   JIT     CARLETONU CS102   SYSTEMS...
BARKER KEN     UMANITOBA 074-438 DATABAS...
BARKER KEN     UMANITOBA 074-452 PROJECT...
BARKER KEN     UOTTAWA   CSI2131 FILE MA...
BARKER KEN     UOTTAWA   CSI4900 PROJECT...
BOYD   SYLVIA  UOTTAWA   CSI5166 COMBINA...
BOYD   SYLVIA  UOTTAWA   CSI4900 PROJECT...
HOLTE  ROBERT  UOTTAWA   CSI1101 COMPUTE...
HOLTE  ROBERT  UOTTAWA   CSI4900 PROJECT...
ROY    DAMIEN  UOTTAWA   MAT1741 ALGÈBRE...
ROY    DAMIEN  UOTTAWA   MAT3543 STRUCTU...
ROY    LANGIS  UOTTAWA   ELG4102 µWAVE &...
MORIN  JOHANNE UQTR       ASY1006 GÉNIE L...
MORIN  JOHANNE UQTR       SIF1016 STRUCTU...
```

N.B. There could be many different index files for `courses.dat`

Indexed Sequential Access

- Records are retrieved from the file either sequentially or directly.
- Record r_i is retrieved by seeking directly to the group containing r_i and then sequentially within the group.
- If the file is indexed on a unique key, access is direct for *every* record.

Q: *What are the advantages of indexed sequential organization?*

A:

Q: *What are the advantages of indexed sequential access?*

A:

Q: *What are the disadvantages of indexed sequential organization?*

A:

Q: *What are the disadvantages of indexed sequential access?*

A: