File Processing

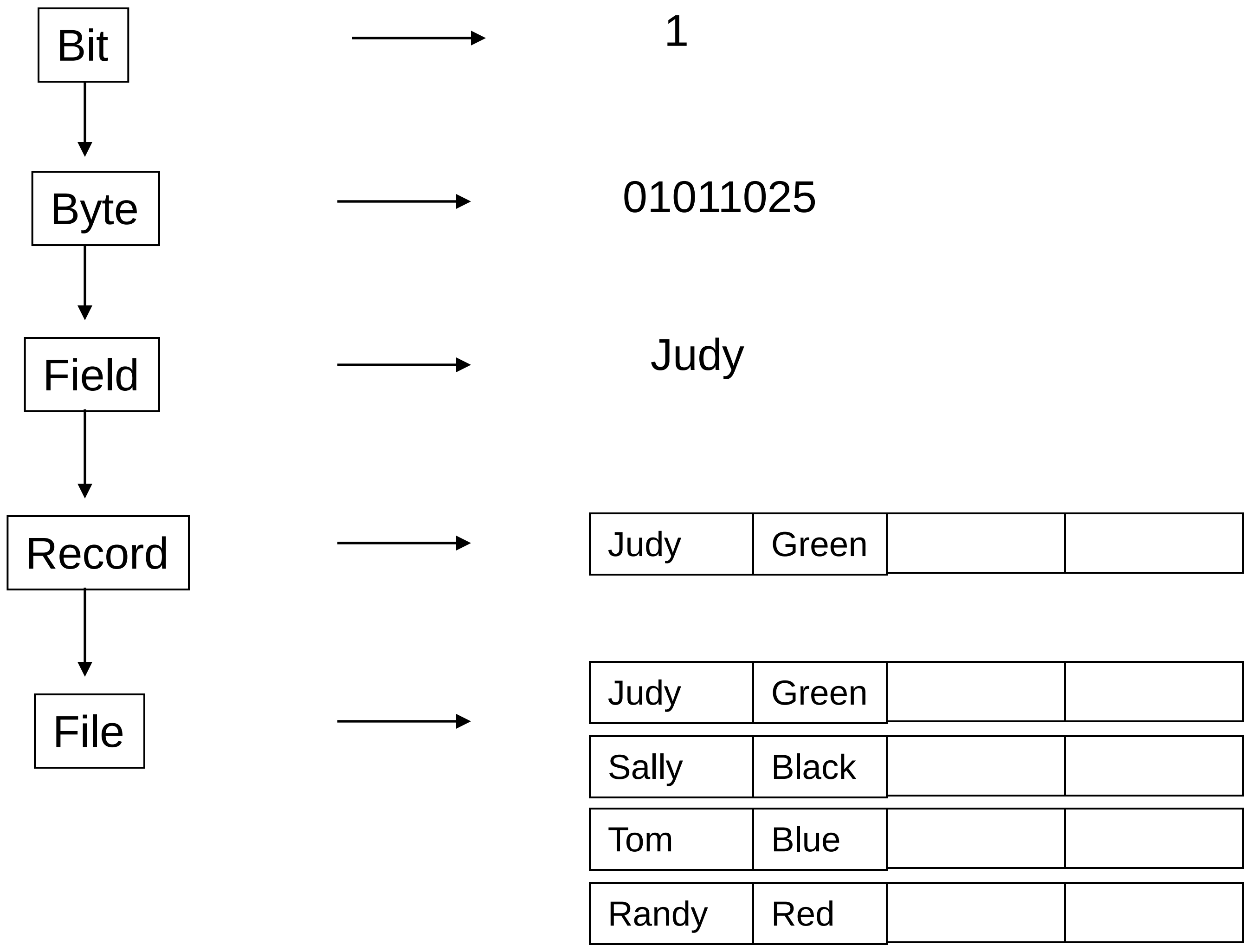
Introduction

- Almost all of the program developed before this is interactive
- In interactive environment, input is via keyboard and output is via screen/monitor
- This type of processing is not suitable if it involves huge amount of input or output to be entered or be displayed on the screen at one time
- Therefore, file processing can solve the problem mentioned

Introduction

- Storage of data in variables and array are temporary.
- File are used for permanent retention of large amount of data.
- Two type of files will be considered ; sequential access file and random access file.

Data Hierarchy



Judy	Green		

Judy	Green		
Sally	Black		
Tom	Blue		
Randy	Red		

Files & Stream

- C views each file simply as a sequential stream of bytes.
- When a file is opened, a stream is associated with the file.
- The files and their associated streams are automatically open when program executions begin, the standard input, the standard output and standard error.
- Stream provides communication channel between files and program.
- For example, standard input stream enable a program to read data from keyboard, the standard output stream enable a program to print data on screen.

Files & Stream

- Opening a file returns a pointer to a FILE structure.
- Standard library provides many functions for reading data and writing data to files.

Creating a sequential file

- Consider the following example :

```
#include<stdio.h>
main () {
int account;
char name[30];
float balance;
FILE *cfPtr;
if ((cfPtr = fopen("clients.txt", "w")) ==
     NULL)
  printf("File cant be opened");
else
  {printf("Enter account, name and
     balance.\n");
   printf("Enter EOF to end input\n");
   printf("?");
   scanf("%d%s%f", &account, name,
     &balance);
while (!feof(stdin)){
   fprintf(cfPtr, "%d %s %.2f\n", account,
   name, balance);
   printf("?");
   scanf("%d%s%f", &account, name,
   &balance);
   }
   fclose(cfPtr);
  }
  return 0;
}
```

Creating a sequential file

- Output
- Enter the account, name and balance.
- Enter the EOF character to end input.
- ? 100   Jones   24.98
- ? 200   Doe   345.67
- ? 300   White   0.00
- ? 400   Stone   -42.16
- ?

Creating a sequential file

- The statement FILE *cfPtr ,
- states that cfPtr is a pointer to a FILE structure .

- The statement if ((cfPtr = fopen("clients.txt", "w")) == NULL),
- names the file "clients.txt" to be used by the program and establish communication with the file.
- The file pointer cfPtr is assigned a pointer to the FILE structure for the file open with fopen(takes two argument, file name & file open mode).

Creating a sequential file

- File mode = r (open file for reading), w(create file for writing), a(append; open or create a for writing at the end of file), r+(open file for update – reading and writing), w+(create file for update), a+(append; open or create file for update)
- If file does not exist, fopen creates that file.

Creating a sequential file

- The statement while(!feof(stdin)),
  - uses function feof to determine whether end-of-file indicator is set for file.
- EOF – for unix system and Mac is <ctrl> d and for IBM PC is <ctrl>z
- The statement fprintf(cfPtr, "%d %s %.2f\n", account, name, balance),
  - writes data to the file clients.dat

Creating a sequential file

- After user enters end-of-file, the program closes the clients.dat with fclose and terminates.

Reading Data from Sequential File

- Data stored in files so that can be retrieved for processing when needed.
- Consider this program

```
#include <stdio.h>
main ()
{
int account;
char name[30];
float balance;
FILE *cfPtr;

if ((cfPtr = fopen("clients.txt", “r")) == NULL)
printf("File cant be opened");
else
{
printf(“%-10s%-13s%s\n”,
“Account”,”Name”, “Balance”);
fscanf(cfPtr, “%d%s%f”,&account, name,
&balance);

while(!feof(cfPtr)) {
printf(“%-10d%-
13s%7.2f\n”,account, name,
balance);
fscanf(cfPtr,
“%d\t\t%s\t\t%f”,&account, name,
&balance);
}
fclose(cfPtr)
}
return 0;
}
```

Reading Data from Sequential File

- Output

```
Account  Name      Balance
100      Jones     24.98
200      Doe       345.67
300      White     0.00
400      Stone     -42.16
```

Random Access File

- In sequential access file, record in a file created with the formatted output function fprintf are not necessarily the same length.
- Individual records of a random access file are normally fixed in length
- This record can be accessed directly without searching through other record. Thus, the searching will be quicker
- It is suitable for the use of airline reservation system, banking system and other kind of transaction processing system.

Random Access File

- Because every record in randomly access file normally fixed in length, data can be inserted in random access file without destroying other data.
- Data stored previously can also be updated or deleted without rewriting the entire file.

Creating a Randomly Accessed File

- Function fwrite is used to transfer a specified numbers of byte beginning at a specified location in memory to file.
- The data is written beginning at the location in the file indicated by the file position pointer.
- Function fread transfer a specified number of bytes from the file specified by the file position to an area in memory with a specified address.

Creating a Randomly Accessed File

- Now, when writing an integer instead of using,
  - fprintf(fPtr, "%d", number)
  - which could print as few as 1 digit or as many as 11 digit, we can use
  - fwrite(&number, sizeof(int), 1, fPtr)
  - which always write 4 bytes from variable number to the file represented by fPtr.

Creating a Randomly Accessed File

- fread is used to read 4 of those bytes into integer variable number.
- The fread and fwrite functions are capable of reading and writing arrays of data to and from disk.
- The third argument of both is the number of element in array that should be read from disk or written to disk.
- The preceding fwrite function call writes a single integer to disk, so third argument is 1.
- File processing program rarely write a single field to a file. Normally, we write one struct at a time.

Creating a Randomly Accessed File

```
#include <stdio.h>
struct clientData{
     int acctNum;
     char lastName[15];
     char firstName[15];
     float balance;};
main(){
int i;
struct clientData blankClient = {0, " ", " ", 0.0};
FILE *cfPtr;
if((cfPtr = fopen("credit.txt", "w")) = = NULL)
     printf("file cant be open");
Else
     { for (I = 1; i<=100; i++)
     fwrite(&blankClient, sizeof(struct ClientData), 1,
      cfPtr);
     }
fclose(cfPtr);
return 0;
 }
```

This program shows how to open a randomly access file, define a record format using struct, write a data to disk, and close the file. This program initialize all 100 records of a file "credit.txt" with empty struct using function fwrite

Writing Data Randomly to a Randomly Accessed File

- #include <stdio.h>
- struct clientData{
  int acctNum;
  char lastName[15];
  char firstName[15];
  float balance;
  };
- main(){
- FILE *cfPtr;
- struct clientData client;
- if((cfPtr = fopen("credit.txt", "r+")) = = NULL)
  - printf("file cant be open");
- else{
  - print("Enter account number(1 to 100, 0 to end input)");
  - scanf("%d", &client.acct.Num);

- while (client.acctNum != 0){
  - printf(“Enter lastname, firstname, balance”);
  - scanf(“%s%s%f, &client.lastName, &client.firstName, &client.balance);
  - fseek(cfPtr, (client.acctNum – 1) * sizeof(struct clientData), SEEK_SET);
  - fwrite(&client, sizeof(struct clientData), 1, cfPtr);
  - printf(“Enter account number”);
  - scanf(“%d”, &client.acctNum);
  - }
- }
- fclose(cfPtr);
- return 0;
- }

Writing Data Randomly to a Randomly Accessed File

- Output

```
Enter account number (1  to 100, 0 to end)
? 29
Enter lastname, firstname, balance
?Brown Nancy -24.54
Enter account number (1  to 100, 0 to end)
? 30
Enter lastname, firstname, balance
?Dunn Stacy 314.33
Enter account number (1  to 100, 0 to end)
? 31
Enter lastname, firstname, balance
?Barker  Doug 0.00
Enter account number (1  to 100, 0 to end)
? 0
```

Writing Data Randomly to a Randomly Accessed File

- The statement fseek(cfPtr, (client.acctNum – 1) * sizeof(struct clientData), SEEK_SET); positions the file position pointer for the file reference by cfPtr to the byte location calculated by (accountNum -1) * sizeof(struct clientData);
- Because of the account number is 1 to 100 but the byte positioning is start from 0, thus the account number need to minus 1.

Reading Data Randomly from a Randomly Accessed File

```
#include <stdio.h>
struct clientData{
    int acctNum;
    char lastName[15];
    char firstName[15];
    float balance;
};
main(){
FILE *cfPtr;
struct clientData client;
if((cfPtr = fopen("credit.txt", "r")) = = NULL)
    printf("file cant be open");
else{
   printf("%-6s%-16s%-11s%10s\n", "Acct", "LastName", " First
    Name", "Balance");
while (!feof(cfPtr)){
  fread(&client, sizeof(struct clientData), 1, cfPtr);
  if (client.acctNum != 0)
     printf(“(“%-6s %16s %11s %10.2f\n ”,
    client.acctNum, client.lastName, client.firstName,
    client.balance);
    }
}
fclose (cfPtr);
 return 0; }
```

Reading Data Randomly from a Randomly Accessed File

- Output

```
Acct     Last Name     First Name     Balance
29       Brown         Nancy          -24.54
30       Dunn          Stacey         314.33
31       Barker        Doug           0.00
```

fread(&client, sizeof(struct clientData), 1, cfPtr);

Reads the number of bytes determined by sizeof(struct clientData) from the file reference by cfPtr and stores the data in the structure client.