Agenda

C File Processing
Files and Streams

- C views each file simply as a sequential stream of bytes. Each file ends either with an end-of-file marker or at a specific byte number recorded in a system-maintained, administrative data structure.

- When a file is opened, a stream is associated with it. Three files and their associated streams are automatically opened when program execution begins, the standard input, the standard output and the standard error. Streams provide communication channels between files and programs.

- For example standard input stream enables a program to read data from the keyboard, and the standard input stream enables a program to read data from the keyboard, and the standard output stream enables a program to print data on the screen.

- Opening a file returns a pointer to a FILE structure (defined in <stdio.h>) that contains information used to process the file. The stanadard input, standard output and standard error are manipulated using file pointers stdin, stdout and stderr.

- The standard library provides many functions for reading data from files and for writing data to files. Function fgetc, like getchar, reads one character from a file. Function fgetc receives as an argument a FILE pointer for the file from which a character will be read. The call fgetc(stdin) reads one character from stdin. This call is equivalent to the call getchar().

- Function fputc, like putchar, writes one character to a file. Function fputc receives as arguments a character to be written and a pointer for the file to which the character will be written. The function cal fputc('a', stdout) writes the character 'a' to stdout. Thic call is equivalent to putchar('a').

C File Processing

Creating a Sequential-Access File

- C imposes no structure on a file. Thus, notions such as a record of a file do not exist as part of the C language. The following example shows how you can impose your own record structure on a file.

```
1  // Creating a sequential file
2  #include <stdio.h>
3
4  int main( void )
5    unsigned int account;
6    char name[ 30 ];
7    double balance;
8    FILE* cfPtr; // cfptr = clients.dat file pointer
9    if( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ){
10     puts( "File could not be opened" );
11   }
12   else{
13     puts( "Enter the account, name and balance." );
14     puts( "Enter EOF to end input." );
15     printf( "%s", "? " );
16     scanf( "%d%29s%lf", &account, name, &balance );
17     while( !feof( stdin ) ){
18       fprintf( "cfPtr, "%d %s %.2f\n", account, name, balance );
19       printf( "%s", "? " );
20       scanf( "%d%29s%lf", &account, name, &balance );
21     }
22   fclose( cfptr );
23   }
24 } // end main
```

- Now let's examine this program. Line FILE *cfPtr states that a cfPtr is a pointer to a FILE structure. A C program administers each file with a separate FILE structure. You need not know the specifics of the FILE structure to use files.

- Each open file must have a separately declared pointer of type FILE that's used to refer the file. Line cfPtr = fopen("clients.dat", "w") names the files to be used by the program and establishes a line of communication with the file. The file pointer cfPtr is assigned a pointer to the FILE structure for the file opened with fopen.

- Function fopen takes two arguments: a file name which can include path information leading to the file's location and a file open mode. The file open mode "w" indicates that the file is to be opened for writing. If a file does not exist and it's opened for writing, fopen creates the file. If an existing file is opened for writing, the contents of the file are discarded without warning.

- In the program, the if statement is used to determine whether the file pointer cfPtr is NULL. If it's NULL, the program prints an error message and terminates. Otherwise, the program processes the input and writes it to the file.

- The program prompts the use to enter the various fields for each record or to enter end-of-file when data entry is complete. The key combination for Linux/MacOS X/UNIX operating systems for entering end-of-file is <Ctrl> d and the key combination for Windows operating system for entering end-of-file is >Ctrl> z.

- Line while(!feof(stdin)) uses function feof to determine whether the end-of-file indicator is set for the file to which stdin refers. The end-of-file indicator informs the program that there's no more data to be processed. The argument to function feof is a pointer to the file being tested for the end-of-file indicator. The function returns a nonzero value when the end-of-file indicator has been set, otherwise the function returns zero.

- Function fprintf is equivalent to printf except that fprintf also receives as an argument a file pointer for the file to which the data will be written. Function fprintf can output data to the stanard output by using stdout as the file pointer.

- After the user enters end-of-file. the program closes the clients.dat file with fclose and terminates. Function fclose also receives the file pointer as an argument. If function fclose is not called explicitly, the operating system normally will close the file when program execution terminates. This is an example of operating system housekeeping.

C File Processing

Creating a Sequential-Access File

- File may be opened in one of several modes. To create a file, or to discard the contents of a file before writing data, open the file for writing (”w”). To read an existing file, open it for reading (”r”).

r	Open an existing file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append: open or create a file for wrting at the end of the file.
r+	Open an existing file for update(reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append: open or create a file for update, writing is done at the end of the file.
rb	Open an existing file for reading in binary mode.
wb	Create a file for writing binary mode. If it already exists, discard the current file.
ab	Append: open or create a file for writing at the end of the file in binary mode.

C File Processing

Reading Data from a Sequential-Access File

- Data is stored in files so that the data can be retrieved for processing when needed. The previous section demonstrated how to create a file for sequential access. The program below reads records from the file clients.dat and prints their contents.

```
1  // Reading a sequential file
2  #include <stdio.h>
3
4  int main( void )
5    unsigned int account;
6    char name[ 30 ];
7    double balance;
8    FILE* cfPtr; // cfptr = clients.dat file pointer
9    if( ( cfPtr = fopen( “clients.dat”, “r” ) ) == NULL ){
10     puts( “File could not be opened” );
11   }
12   else{
13     printf( “%-10s%-13s%s\n”, “Account”, “Name”, “Balance” );
14     fscanf( cfPtr, “%d%29s%lf”, &account, name, &balance );
15     while( !feof( cfPtr ) ){
16       printf( ”%-10d-13s%7.2f\n”, account, name, balance );
17       fscanf( cfPtr, “%d%29s%lf”, &account, name, &balance );
18     }
19   fclose( cfptr );
20   }
21 } // end main
```

Reading Data from a Sequential-Access File

- To retrieve data sequentially from a file, a program normally starts reading from the beginning of the file and reads all data consecutively until the desired data is found. It may be desirable to process the data sequentially in a file several times during the execution of a program. The statement

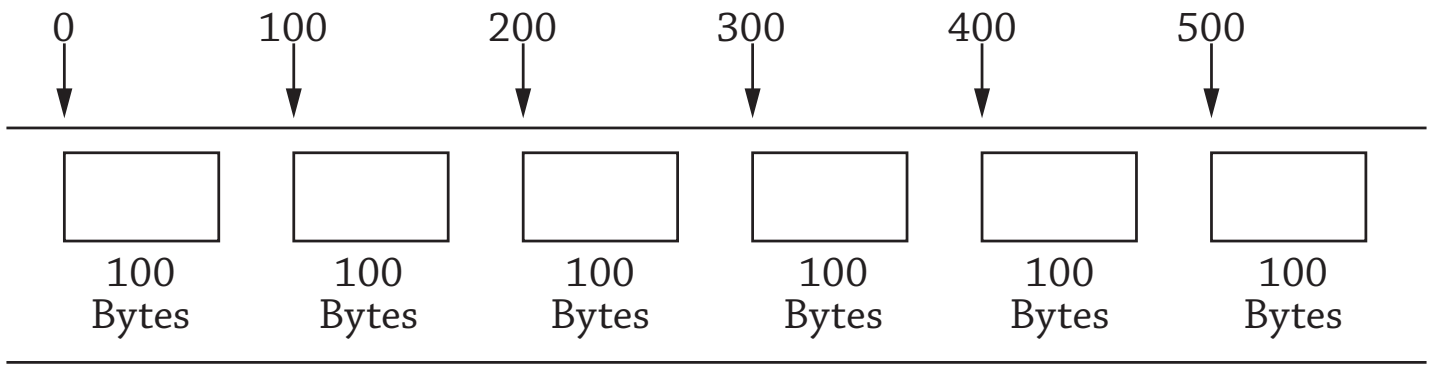
  rewind(cfPtr);

  causes a program's file position pointer to be repositioned to the beginning of the file pointed to by cfPtr. The file position pointer is not really a pointer. Rather it's an integer value that specifies the byte in the file at which the next read or write it to occur. This sometimes referred to as the file offset.

Creating a Random-Access File

- As we stated previously, records in a file created with the formatted output function fprintf are not necessarily the same length. However, individual records of a random access file are normally fixed in length and may be accessed directly without searching through other records.

- Because every record in a random-access file normally has the same length, the exact location of a record relative to the beginning of the file can be calculated as a function of the record key. Figure below illustrates one way to implement a random-access file. Such a file is like a greight train with many cars, some empty and some with cargo. Each car in the train has the same length.



- Fixed-length records enable data to be inserted in a random-access file without destroying other data in the file. Data stored previously can also be updated or deleted without rewriting the entire file.

Creating a Random-Access File

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

- Although fread and fwrite read and write data, such as integers, i fixed-size rather than variable-size format, the data they handle are processed in computer raw data format (bytes of data) rather than in printf's and scanf's human-readable text format.

C File Processing

Creating a Random-Access File

- The program below initializes all 100 records of the file credit.dat with empty structs using the function fwrite. Each empty struct contains 0 for the account number, empty string for the last name and the first name and 0.0 for the balance.

```
1  // Creating a random-access file
2  #include <stdio.h>
3
4  struct clientData{
5     unsigned int acctNum;
6     char lastName[ 15 ];
7     char firstName[ 10 ];
8     double balance;
9  };
10 int main( void )
11    unsigned int i;
12    struct clientData blankClient = { 0, “”, “”, 0.0 };
13    FILE *cfPtr;
14
15    if( ( cfPtr = fopen( “clients.dat”, “wb” ) ) == NULL ){
16       puts( “File could not be opened” );
17    }
18    else{
19       for( i = 1; i <= 100; i++ ){
20          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
21       }
22    fclose( cfptr );
23    }
24 } // end main
```

Writing Data Randomly to a Random-Access File

- The program below writes data to the file credit.dat. It uses the combination of fseek and fwrite to store data at specific locations in the file. Function fseek sets the file position pointer to a specific position in the file, then fwrite writes the data.

```
1  // Creating a random-access file
2  #include <stdio.h>
3
4  struct clientData{
5     unsigned int acctNum;
6     char lastName[ 15 ];
7     char firstName[ 10 ];
8     double balance;
9  };
10
11 int main( void )
12    unsigned int i;
13    struct clientData client = { 0, "", "", 0.0 };
14    FILE *cfPtr;
15
16    if( ( cfPtr = fopen( "clients.dat", "rb+" ) ) == NULL ){
17       puts( "File could not be opened" );
18    }
```

```
19    else{
20       printf( "%s", "Enter accout number\n" );
21       scanf( "%d", &client.acctNum );
22       while( client.acctNum != 0 ){
23          printf( "%s", "Enter last name, first name, balance\n?" );
24          fscanf( stdin, "%14s%9s%lf", client.lastName, client.firstName, &client.balance );
25          fseek( cfPtr, ( client.acctNum - 1 ) * sizeof( struct clientData ), SEEK_SET );
26          fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
27          printf( "%s", "Enter accout number\n" );
28          scanf( "%d", &client.acctNum );
29       }
30       fclose( cfPtr );
31    }
32 } // end main
```

- The function prototype for fseek is

int fseek(FILE* stream, long int offset, int whence);

where offset is the number of bytes to seek from whence in the file pointed to by stream, a positive offset seeks forward and a negative one seeks backward. Argument whence is one of the values SEEK_SET, SEEK_CUR or SEEK_END which indicate the location from whick the seek begins. SEEK_SET indicates that the seek starts at the beginning of the file, SEEK_CUR indicates that the seek starts at the current location in the file, and SEEK_END indicates that the seek starts at the end of the file.

C File Processing
Reading Data from a Random-Access File

- The program below reads sequentially every record in the credit.dat file determines whether each record contains data and displays the formatted data for records containing data.

```
1  // Reading a random-access file
2  #include <stdio.h>
3  struct clientData{
4     unsigned int acctNum;
5     char lastName[ 15 ];
6     char firstName[ 10 ];
7     double balance;
8  };
9  int main( void )
10    unsigned int i;
11    struct clientData client = { 0, “”, “”, 0.0 };
12    FILE *cfPtr;
13    int result;
14    if( ( cfPtr = fopen( “clients.dat”, “rb” ) ) == NULL ){
15       puts( “File could not be opened” );
16    }
17    else{
18       printf( “%-6s%-16s%-11s%10s\n”, “Acct”, “Last Name”, “First Name”, “Balance” );
19       while( !feof( cfPtr ) ){
20          result = fread( client, sizeof( struct clientData ), 1, cfPtr );
21          if( result != 0 && client.acctNum != 0 )
22          printf( “%-6s%-16s%-11s%10.2f\n”, client.acctNum, client.lastName, client.firstNaeme, cl
23       }
24    fclose( cfptr );
25    }
26 } // end main
```