Chapter 11: File Processing

Introduction

* Storage of data in variables and arrays is temporary; all such data is lost when a program terminates
* Files are used for permanent retention of large amounts of data since they are stored on secondary storage devices, especially disk storage devices.

Data Hierarchy

* bit: the smallest data item in a computer, either "0" or "1"
* character: digits, letters, and special symbols; represented as a pattern of
bits; commonly composed of eight bits
* field: a group of characters that conveys meaning
* record (struct): composed of several related fields
* file: a group of related record
* to facilitate the retrieval of specific records from a file, at least one field in each record is chosen as a record key
* a record key identifies a record as belonging to a particular entity.
* database: a group of related files

Files and Streams

* C views each file simply as a seqiential stream of bytes.
* Each file ends with an "end-of-file" (EOF) marker.
* When a file is opened, a stream is associated with the file.
* Three file are automatically opened:
    0. standard input (stdin): data from keyboard
    1. standard output (stdout): data to screen
    2. standard error (stderr): data to error device (usually screen)

* Opening a file returns a pointer to a "FILE" structure (defined in <stdio.h>) that contains information used to process the file.

* Function "fgetc", which receives as an argument a "FILE" pointer, reads one character from that file.
* "fgetc(stdin)" is equivalent to "getchar()".
* Function "fputc" receives as arguments a character to be written and a pointer for the file to which the character will be written.
* "fputc('a', stdout)" is equivalent to "putchar('a')"

* "fgets" and "fputs" can be used to read a line from a file and write a line to a file, respectively, similar to "gets" and "puts" for "stdin" and "stdout".

Sequential Access Files

* C imposes no structure on a file.
* E.g.

```
        #include <stdio.h>

        main()
        {
           int account;
           char name[30];
           float balance;
           FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

           if ((cfPtr = fopen("clients.dat", "w")) == NULL)
              printf("File could not be opened\n");
           else {
              printf("Enter the 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;
        }
```

* file position pointer - indicating the number of the next byte in the file to be read or written
* FILE *fd;
* C program administers each file with a separate "FILE" structure.
* fd = fopen("file.txt","r");
* Function "fopen(..)" takes two arguments: a file name and a file open mode.
* modes:
    "r" - open for reading
    "w" - create or erase for writing
    "a" - append for writing at the end of file
    "r+" - open for update (reading and writing)
    "w+" - create or erase for update
    "a+" - append for update at the end of file
* If an error occurs, "fopen" returns "NULL".
* feof(fd); to determine whether the end-of-file indicator is set for that file.
* The end-of-file indicator informs the program that there is no more data to be processed.
* fprintf(fd, "%d %s %d\n", id, name, balance);
* Function "fprintf" is equivalent to "printf" except that "fprintf" also receives as argument a file pointer for the file to which the data will be written.
* fclose(fd);
* "fclose(..)" the file of file pointer as an argument.
* If function "fclose" is not called explicitly, the OS normally will close the file when the program execution terminates.

Reading Data from a Sequential Access File
----
* E.g.

```
        #include <stdio.h>

        main()
        {
```

```
        int account;
        char name[30];
        float balance;
        FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

        if ((cfPtr = fopen("clients.dat", "r")) == NULL)
           printf("File could not be opened\n");
        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%s%f", &account, name, &balance);
           }

           fclose(cfPtr);
        }

        return 0;
     }
```

* fscanf(fd, "%d %s %d\n", &x, name, &z);
* Function "fscanf" is equivalent to "scanf" except that "fscanf" also receives as argument a file pointer for the file from which the data will be read.

* 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 are found.
* rewind(fd); causes a program's file position pointer to be repositioned to the beginning of the file pointed by "fd"

* E.g.

```
     #include <stdio.h>

     main()
     {
        int request, account;
        float balance;
        char name[30];
        FILE *cfPtr;

        if ((cfPtr = fopen("clients.dat", "r")) == NULL)
           printf("File could not be opened\n");
        else {
           printf("Enter request\n"
                  " 1 - List accounts with zero balances\n"
                  " 2 - List accounts with credit balances\n"
                  " 3 - List accounts with debit balances\n"
                  " 4 - End of run\n? ");
           scanf("%d", &request);

           while (request != 4) {
              fscanf(cfPtr, "%d%s%f", &account, name, &balance);

              switch (request) {
                 case 1:
                    printf("\nAccounts with zero balances:\n");
```

```
                        while (!feof(cfPtr)) {

                           if (balance == 0)
                              printf("%-10d%-13s%7.2f\n",
                                     account, name, balance);

                           fscanf(cfPtr, "%d%s%f",
                                  &account, name, &balance);
                        }

                        break;
                     case 2:
                        printf("\nAccounts with credit balances:\n");

                        while (!feof(cfPtr)) {

                           if (balance < 0)
                              printf("%-10d%-13s%7.2f\n",
                                     account, name, balance);

                            fscanf(cfPtr, "%d%s%f",
                              &account, name, &balance);
                        }

                        break;
                    case 3:
                        printf("\nAccounts with debit balances:\n");

                        while (!feof(cfPtr)) {
                           if (balance > 0)
                              printf("%-10d%-13s%7.2f\n",
                                     account, name, balance);

                           fscanf(cfPtr, "%d%s%f",
                                  &account, name, &balance);
                        }

                        break;
                   }

                   rewind(cfPtr);
                   printf("\n? ");
                       scanf("%d", &request);
         }

         printf("End of run.\n");
         fclose(cfPtr);
      }

      return 0;
   }
```

* sequential file cannot be modified without the risk of destroying other data in the file.
* e.g. The record for "White" was written to the file as

        300 White 0.00

* If the record is rewritten beginning at the same location in the file using the new name, the record become,

```
        300 Worthington 0.00
```

* New record is larger than the original record. The characters beyond the second "o" in "Worthington" would overwrite the beginning of the next sequential record in the file.
* sequential access with "fprint" and "fscanf" is not usually used to update records in place, but the entire file is usually rewritten.

Random Access Files
----
* individual records are fixed in length
* may be accessed directly without searching through other records
* the exact location of a record relative to the beginning of the file can be calculated as a function of the record key
* data can be inserted in a randomly accessed file without destroying other data in the file

Creating a Randomly Accessed File
----
* Function "fwrite" transfers a specified number of bytes beginning at a specified location in memory to a file.
* 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 specific address.
* Compare:

```
    1. fprintf(fPtr, "%d", number);
    2. fwrite(&number, sizeof(int), 1, fPtr);
```

* The data handled by "fread" and "fwrite" is processed in computer "raw data" format (i.e. bytes of data) rather than in human-readable format.

* file processing programs rarely write a single field to a file; they write one "struct" at a time
* E.g.

```
        #include <stdio.h>

        struct clientData {
           int acctNum;
           char lastName[15];
           char firstName[10];
           float balance;
        };

        main()
        {
           int i;
           struct clientData blankClient = {0, "", "", 0.0};
           FILE *cfPtr;

           if ((cfPtr = fopen("credit.dat", "w")) == NULL)
              printf("File could not be opened.\n");
           else {

              for (i = 1; i <= 100; i++)
                 fwrite(&blankClient,
                        sizeof(struct clientData), 1, cfPtr);

              fclose (cfPtr);
           }
           return 0;
        }
```

```
* "fwrite(...)" writes a block of data to a file
* "&blankClient" is the address of block
* "sizeof(struct clientData)" is the size of block in byte
* "1" is the number of block to write
* "cfPtr" is the file pointer
* e.g. fwrite(&number, sizeof(int), 1, fPtr);
* Function "fwrite" can actually be used to write several elements of
an array of objects.
* To write several array elements, the programmer supplies a pointer
to an array as the first argument in the call to "fwrite", and
specifies the number of elements to be written as the third argument.
```

```
Writing Data Randomly to a Randomly Accessed File
-------------------------------------------------
* E.g.
        #include <stdio.h>

        struct clientData {
           int acctNum;
           char lastName[15];
           char firstName[10];
           float balance;
        };

        main()
        {
           FILE *cfPtr;
           struct clientData client;

           if ((cfPtr = fopen("credit.dat", "r+")) == NULL)
              printf("File could not be opened.\n");
           else {
              printf("Enter account number"
                     " (1 to 100, 0 to end input)\n? ");
              scanf("%d", &client.acctNum);

              while (client.acctNum != 0) {
                 printf("Enter lastname, firstname, balance\n? ");
                 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\n? ");
                 scanf("%d", &client.acctNum);
              }
           }

           fclose(cfPtr);

           return 0;
        }

* "fseek(...)" sets the file position pointer to a specific position
in the file
```

* "cfPtr" is the file pointer
* "(client.acctNum - 1) * sizeof(struct clientData)" is the offset or the displacement
* "SEEK_SET" indicates that the file position pointer is positioned relative to the beginning of the file by the amount of the offset
* "SEEK_CUR" indicates that the seek starts at the current location in the file
* "SEEK_END" indicates that the seek starts at the end of the file

Reading Data Randomly from a Random Accessed File
----

* E.g.

```
        #include <stdio.h>

        struct clientData {
           int acctNum;
           char lastName[15];
           char firstName[10];
           float balance;
        };

        main()
        {
           FILE *cfPtr;
           struct clientData client;

           if ((cfPtr = fopen("credit.dat", "r")) == NULL)
              printf("File could not be opened.\n");
           else {
              printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
                     "First Name", "Balance");

              while (!feof(cfPtr)) {
                 fread(&client, sizeof(struct clientData), 1, cfPtr);

                 if (client.acctNum != 0)
                    printf("%-6d%-16s%-11s%10.2f\n",
                           client.acctNum, client.lastName,
                           client.firstName, client.balance);
              }
           }

           fclose(cfPtr);

           return 0;
        }
```

* "fread(...)" reads a specified number of bytes from a file
* "&client" is the struct variable to store the read data
* "sizeof(struct clientData)" is the size of block in byte
* "1" is the number of block to be read
* "cfPtr" is the file pointer

Exercise
========

1. Write the "selectRecord" program to read a particular record only after the user input the account number.

2. Combine "Create", "Write", "Report" and "Search" into a program and use a menu for user to choose. "Report" should also generate a file report.