第１０回(6/25)

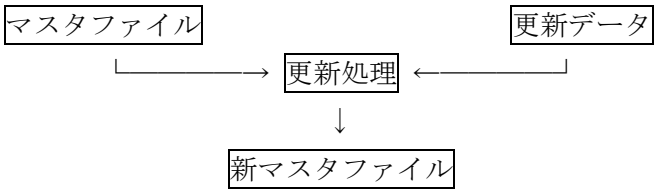
３．ファイルとその応用

(3) ファイルの更新

《シーケンシャルファイルの更新》

シーケンシャルファイルでは，各レコードが可変長で連続して格納されており，その中の特定のレコードを変更することができない。そこで一般的には，マスタファイルからデータを取り出し，更新処理を行ったあとに新マスタファイルに書き込む。

注）マスタファイル：主ファイル，基本ファイルと呼ばれるファイルで内容は比較的固定的であり，最も基本となるファイルのこと。また，マスタファイルを更新したものを新マスタファイルという。これに対して，データ処理の中間結果や，変更データを集めて記録したファイルをトランザクションファイルという。



例題３－７

例題３－３によって作成したデータファイル name.txt 内の任意のデータを変更して，新しいデータファイル nameout.txt に保存するプログラムをつくりなさい。

▼出力結果

データを画面に表示します。

1　　Kyoko Ishida
2　　Ichiro Suzuki
3　　Yoshiko Tanaka
4　　Makoto Sasaki
5　　Yoko Kobayashi

何番のデータを変更しますか　：2↵
新しいデータを入力して下さい　：Saburo Suzuki↵

考え方

例題３－４と同様に，すべてのデータを画面に表示する。そして，その中から変更の必要なデータを指定するようにし，name.txt をマスタファイル，nameout.txt を新マスタファイルとして考えるとよい。

▼プログラム３－７

```
01    /* E3-7 */
02    /*シーケンシャルファイルの更新*/
03    #include <stdio.h>
04    #include <stdlib.h>
05
06    main()
07    {
08
09        FILE *fp1, *fp2;
10        char last_name[15], first_name[15];
11        char last_newname[15], first_newname[15];
12        int i, n, count;
13
14        if ((fp1 = fopen("name.txt", "r")) == NULL) {
15            printf("ファイルがオープンできません。¥n");
16            exit(1);
17        }
18        if ((fp2 = fopen("nameout.txt", "w")) == NULL) {
19            printf("ファイルがオープンできません。¥n");
20            exit(1);
21        }
22        count = 0;
23        printf("データを画面に表示します。 ¥n");
24        while (fscanf(fp1, "%s%s", first_name, last_name) != EOF) {
25            printf("%2d¥t%s %s¥n", count+1, first_name, last_name);
26            count++;
27        }
28        printf("何番のデータを変更しますか :");
29        scanf("%d", &n);
30        printf("新しいデータを入力して下さい :");
31        scanf("%s%s", first_newname, last_newname);
32
33        fseek(fp1, 0L, SEEK_SET);   /* または rewind(fp1);  */
```

```
34
35        for (i = 0; i < count; i++) {
36            fscanf(fp1, "%s%s", first_name, last_name);
37            if (i == n-1) {
38                fprintf(fp2, "%s %s", first_newname, last_newname);
39                fputc('¥n', fp2);
40            } else {
41                fprintf(fp2, "%s %s", first_name, last_name);
42                fputc('¥n', fp2);
43            }
44        }
45
46        fclose(fp1);
47        fclose(fp2);
48    }
```

練習問題２９

１．例題３－７によって作成したファイル nameout.txt の中から任意のデータを削除し，新ファイル namedel.txt に保存するプログラムをつくりなさい。すなわち，namedel.txt に格納される氏名の数は１つ減少します。最後に，namedel.txt のデータを画面に表示しなさい。

《ヒント》namedel.txt に書き込んでから，最後に画面に表示するには読み出しが必要である。このため，最初に namedel.txt のオープンするときにモードを"w+"と指定することに注意する。

▼出力例

データを画面に表示します。

1　　Kyoko Ishida
2　　Saburo Suzuki
3　　Yoshiko Tanaka
4　　Makoto Sasaki
5　　Yoko Kobayashi

何番のデータを削除しますか　：2↵

削除後のデータを画面に表示します。

1　　Kyoko Ishida
2　　Yoshiko Tanaka
3　　Makoto Sasaki
4　　Yoko Kobayashi

《ランダムファイルの更新》

ランダムファイルでは固定長でレコードが格納されているため，fseek 関数を利用することで任意の位置のデータを更新することができ，シーケンシャルファイルのように新たにファイルを作成する必要がない。

例題３－８

例題３－５によって作成したデータファイル name2.txt 内の任意のデータを変更し，もとのデータファイル name2.txt に書き戻すプログラムをつくりなさい。

▼出力結果３－８

```
1       Kyoko Ishida
2       Ichiro Suzuki
3       Yoshiko Tanaka
4       Makoto Sasaki
5       Yoko Kobayashi
何番のデータを変更しますか　：2↵
新しいデータを入力して下さい　：Saburo Suzuki↵
```

考え方

同一のファイルに対して読み書きを行うので，ファイルを読み書き可能なモードでオープンする。

▼プログラム３－８

```
01      /* E3-8 */
02      /*ランダムファイルの更新*/
03      #include <stdio.h>
04      #include <stdlib.h>
05
06      main()
07      {
08          FILE *fp1;
09          char last_name[15], first_name[15];
10          int i = 0 , n;
11
12          if ((fp1 = fopen("name2.txt", "r+")) == NULL) {
13              printf("ファイルがオープンできません。¥n");
14              exit(1);
15          }
16
17          while(1) {
18              fseek(fp1, i * 30L, SEEK_SET);
19              if (fscanf(fp1, "%s%s", first_name, last_name) == EOF) break;
20              printf("%2d¥t%s %s¥n", i+1, first_name, last_name);
21              i++;
22          }
23
24          printf("何番のデータを変更しますか :");
25          scanf("%d", &n);
26          printf("新しいデータを入力して下さい :");
27          scanf("%s%s", first_name, last_name);
28          fseek(fp1, (n-1) * 30L, SEEK_SET);
29          fprintf(fp1, "%15s%15s", first_name, last_name);
30
31          fclose(fp1);
32      }
```

練習問題２９

２．例題３－８によって作成したファイル name2.txt の中から任意のデータを削除するプログラムをつくりなさい。但し，削除データには，fprintf 関数で書式指定"%15s%15s"により 14 個の*を並べた"**********"を 2 つ上書きすることとする。最後に，削除後のデータを表示しなさい。

▼出力例

データを画面に表示します。

1　　Kyoko Ishida
2　　Saburo Suzuki
3　　Yoshiko Tanaka
4　　Makoto Sasaki
5　　Yoko Kobayashi

何番のデータを削除しますか　：2↵

削除後のデータを画面に表示します。

1　　Kyoko Ishida
2　　********** **********
3　　Yoshiko Tanaka
4　　Makoto Sasaki
5　　Yoko Kobayashi

３．練習問題２８－１によって作成した電話帳のファイルを自由に変更することのできるプログラムをつくりなさい。すなわち，出力例にあるように，変更したいデータの位置番号を指定して，そこのデータ内容を新たにキーボードから入力した氏名と電話番号に変更できるようにします。最後に，変更した後のデータを画面に表示しなさい。

▼出力例

```
データを画面に表示します。
    1       Ichiro Suzuki      03-1111-2222
    2       Hanako Yamato      042-387-5555
    3       Hidetoshi Nakata   045-333-1111
何番のデータを変更しますか   ：2↵

新しい名前を入力して下さい。
Hideki Matsui↵
新しい電話番号を入力して下さい。
211-234-9999↵

変更後のデータを画面に表示します。
    1       Ichiro Suzuki      03-1111-2222
    2       Hideki Matsui      211-234-9999
    3       Hidetoshi Nakata   045-333-1111
```

(4) バイナリデータの入出力方法

fprintf(), fscanf()は，数値データをファイルに読み書きする方法としては必ずしも最適ではない。それは，

バイナリフォーマット　⇔　ASCII テキスト

の相互変換を行うからである。例えば，fprintf()を使用して数字を書き込む場合，数字はバイナリフォーマットから ASCII テキストに変換される。逆に，fscanf()を使用して数字を読み込む場合，数字を ASCII テキストからバイナリフォーマットに変換しなければならない。この変換処理は，処理時間やファイル容量で問題になる場合がある。

このため，C言語には，バイナリフォーマットのままで，あらゆる型のデータの読み書きを実行する関数 fread(), fwrite()が用意されている。書式は次の通り。

```
size_t fread(void *BUF, size_t SIZE, size_t NUM, FILE *fp);
size_t fwrite(void *BUF, size_t SIZE, size_t NUM, FILE *fp);
```

《汎用ポインタ void * の利用》

データの読み書きに使用するバッファ BUF が汎用ポインタ void * で宣言されている点に注意する。これにより，あらゆる型のデータの読み書きが可能となる。

fread()関数は，

fp で関連づけられたファイルから，
SIZE バイトの大きさのオブジェクトを
NUM 個読み込んで
BUF が指し示すバッファに格納する。
関数の戻り値は，実際に読み込んだオブジェクトの個数となる。

fwrite()関数は，

BUF が指し示すバッファから，
SIZE バイトの大きさのオブジェクトを
NUM 個読み出して
fp で関連づけられたファイルに書き込む。
関数の戻り値は，実際に書き込んだオブジェクトの個数となる。

size_t 型：ヘッダファイル stdlib.h で定義されている。この型の変数は，コンパイラがサポートする最大オブジェクトの大きさを持つ値を保持できる。unsigned または unsigned long と考えてよい。移植性を高めるために用いる。

例題３－９

10 個の要素を持つ配列を浮動小数点数で埋め，それらをバイナリ形式の内部表現のままで，ファイルに書き出して，再び読み込むプログラムをつくりなさい。

考え方

fwrite()，fread()を用いる。その際，バイナリデータで入出力操作を行うため，ファイルはバイナリ演算用のモードで開く必要がある。

▼プログラム３－９

```
01      /* E3-9 */
02      /* バイナリデータの入出力 */
03      #include <stdio.h>
04      #include <stdlib.h>
05
06      double d[10] = {
07          10.23, 19.87, 1002.23, 12.9, 0.897,
08          11.45, 75.34, 0.0, 1.01, 875.875
09      };
10
11      int main()
12      {
13          int i;
14          FILE *fp;
15
16          if ((fp = fopen("myfile", "wb")) == NULL) {
17              printf("ファイルを開くことができません¥n");
18              exit(1);
19          }
20
21          for (i = 0; i < 10; i++)
```

```
22              if (fwrite(&d[i], sizeof(double), 1, fp) != 1) {
23                  printf("書き込みエラー¥n");
24                  exit(1);
25              }
26          fclose(fp);
27
28          if ((fp = fopen("myfile", "rb")) == NULL) {
29              printf("ファイルを開くことができません¥n");
30              exit(1);
31          }
32
33          /* 配列をクリアする*/
34          for (i = 0; i < 10; i++) d[i] = -1.0;
35
36          for (i = 0; i < 10; i++)
37              if (fread(&d[i], sizeof(double), 1, fp) != 1) {
38                  printf("読み込みエラー¥n");
39                  exit(1);
40              }
41          fclose(fp);
42
43          /* 配列を表示する*/
44          for (i = 0; i < 10; i++) printf("%f ", d[i]);
45          printf("¥n");
46          return 0;
47      }
```

例題３－１０

例題３－９では，fwrite()，fread()をそれぞれ 10 回呼んでいる。これを，それぞれ 1 回ずつしか呼び出さないようにプログラムを作り替えなさい。

考え方

fwrite(), fread()を用いる際，配列全体を１回の操作で書き出したり，読み出したりすればよい。sizeof 演算子を活用する。

▼プログラム３－１０

```
01      /* E3-10 */
02      /* バイナリデータの入出力 */
03      #include <stdio.h>
04      #include <stdlib.h>
05
06      double d[10] = {
07          10.23, 19.87, 1002.23, 12.9, 0.897,
08          11.45, 75.34, 0.0, 1.01, 875.875
09      };
10
11      int main()
12      {
13          int i;
14          FILE *fp;
15
16          if ((fp = fopen("myfile", "wb")) == NULL) {
17              printf("ファイルを開くことができません¥n");
18              exit(1);
19          }
20
21          /* 配列全体を一度に書き込む */
22          if (fwrite(d, sizeof d, 1, fp) != 1) {
23              printf("書き込みエラー¥n");
24              exit(1);
25          }
26          fclose(fp);
27
28          if ((fp = fopen("myfile", "rb")) == NULL) {
```

```
29              printf("ファイルを開くことができません.¥n");
30              exit(1);
31          }
32
33          /* 配列をクリアする */
34          for (i = 0; i < 10; i++) d[i] = -1.0;
35
36          /* 配列全体を一度に読み込む */
37          if (fread(d, sizeof d, 1, fp) != 1) {
38              printf("読み込みエラー¥n");
39              exit(1);
40          }
41          fclose(fp);
42
43          /* 配列を表示する */
44          for (i = 0; i < 10; i++) printf("%f ", d[i]);
45          printf("¥n");
46          return 0;
47      }
```

練習問題２９

４．（難問）個人レコード（氏名と電話番号）を構造体で定義して，このレコード単位にバイナリデータで読み書きすることにより，電話帳のランダムファイルを作成してみましょう。下記の出力例のように動作させます。格納したデータを読み出す際は，総レコード数分のメモリ領域を確保して一括読み込みを行って下さい。但し，個人レコードの構造体宣言は次の通り。

```
typedef struct {
    char first_name[15];
    char last_name[15];
    char tel_no[15];
} Personal;
```

▼出力例

```
個人データを入力してください（終了は Ctrl-d）

Name ： Ichiro Suzuki↲
Tel  ： 03-1234-5678↲

Name ： Hideki Matsui↲
Tel  ： 03-9999-1111↲

Name ： Ctrl-d

データをファイルに格納しました！
格納したデータを読み出すと次の通り：

Ichiro Suzuki
03-1234-5678

Hideki Matsui
03-9999-1111
```

《ヒント》個人データのファイル name3 は，読み書きできるバイナリファイルとして，アクセスモードは"w+b"でオープンする。作成されたバイナリファイル name3 の内容をバイト単位で確認するには，8 進ダンプを次のように行ってみよう。但し，コマンド od のオプション -cb は調べてみること。

```
$od -cb name3↵
```

また，総レコード数分のメモリ領域を確保するには，動的なメモリ確保を行うライブラリ関数 malloc を活用しなさい。