計算機言語 II 第 10 回 バイナリファイル

http://www.math.u-ryukyu.ac.jp/~suga/gengo/2018-2/10.pdf

1 バイナリ (binary) ファイル

コンピュータ関連の講義で何度も述べていますが, ファイルには, 文字列データしか入っていないテキストファイル (例えば, LaTeX や C のソース) と, それ以外のデータも含むファイル (画像データ, dvi ファイル, この講義て作る a.out) があり, 後者のことをバイナリファイルと呼びます.

C のようなコンパイラ系言語を利用すると, ファイルに対してバイナリデータを直接読み書きすることができます. そのためには, ファイルを開く際に, モードに対して「b」の文字を付加します.

```
fp = fopen(hoge, "rb");  /* hoge をバイナリファイルとして読み込みモードで開く */
fp = fopen(hogehoge, "wb");  /* hogehoge をバイナリファイルとして書き込みモードで開く */
...
```

バイナリモードで開いたファイルに対する読み書きは, `fread()`, `fwrite()` を利用します. それぞれ, `man 3 fread`, `man 3 fwrite` でその使い方がわかります.

`fread()`, `fwrite()` は `size_t` 型を返す関数で, それぞれ, 読み込み, 書き込みに成功した数を返します.

`fread()` は, その時点でのファイルポインタが指す場所 (特に指定しなければ, 一番最初の呼び出しではファイルの先頭) から与えられたデータ量だけファイルの内容を読み, 読んだ分だけファイルポインタをずらします. すなわちファイルポインタは, 単純なファイルへのポインタではなく, ライブラリ関数がファイルのどの部分から読むかという情報も保持している構造体へのポインタなのです. `fwirte()` におけるファイルポインタの扱いも同様です.

`fread()` では, 読み込んだ値を第一引数のポインタ値が指すメモリに順に格納 (コピー) します. プログラマは, 読み込みデータ量に見合うメモリをあらかじめ確保する必要があります.

`fread()`, `fwrite()` でファイルの途中から読み書きをするには, `fseek()` というライブラリ関数を用いて, ファイルポインタの指す場所を変更させます.

また, 途中まで読み書きしたファイルを, また先頭から扱うことも可能で, `rewind()`(なぜか先頭に f が付かない) というライブラリ関数を用います.

教科書 p. 352, List 13–10 は, ファイル中身を 16 進表示するプログラムで, この種のソフトは, ハッカーによく利用されているものです.

ただし, 標準出力に結果を出し, それ以外は標準エラー出力に出すといった C の作法が守れられていないので, 下のように書き直しました. ファイルが存在しない時には, 実行環境に `EXIT_FAILURE` という値を返してプログラムを終了することにし, それをライブラリ関数 `exit()` で実現しています.

```
/* list 1310m.c */

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    unsigned count = 0;
    unsigned char buf[16];
    FILE *fp;
    char filename[FILENAME_MAX];

    fprintf(stderr, "ファイル名: ");
    scanf("%s", filename);

    if ((fp=fopen(filename, "rb"))== NULL){
        fprintf(stderr, "ファイル %s を開けません\n", filename);
        exit(EXIT_FAILURE);
    } else {
        while((n=fread(buf, sizeof(char), 16,fp))>0){
            int i;
            printf("%08lX ", count);

            for (i=0; i < n; i++){
                printf("%02X ", (unsigned)buf[i]);
            }

            if (n < 16){
                for (i=n; i< 16; i++){
                    printf("   ");
                }
            }

            for (i =0; i < n; i++){
                putchar(isprint(buf[i])? buf[i]: '.');
            }

            putchar('\n');
            count+=16;
        }
        fclose(fp);
    }
    return 0;
}
```

バイナリデータの読み書きに対する注意

上のプログラムでは, `fread()` を用いてバイナリモードで開いたファイルを読んでいます. しかし, 画像ファイルのような場合, 多くはバイト単位での読み書きで意味のある動作をさせることも可能です. 普通のテキストモードで開いて, `fgetc()`, `fputc()` というライブラリ関数で, 1 バイト単位で読み書きをするのです. これから参照する書籍,

奥村晴彦著, 改訂新版 C 言語によるアルゴリズム辞典, 技術評論社

でも画像ファイルを作るときに, `fputc()` を用いてバイト単位で書き出しています.

2 線形代数をプログラムする

1 年次の授業で線形代数を学習しましたが, その中で応用上最も重要なのは, 「連立 1 次方程式を解く」です. これ以外にも固有値の計算や, 逆行列の計算なども重要です. 連立方程式の解を求めたり, 逆行列を求める計算は, 線形代数で学習した「行列の行基本変形を用いた掃き出し」が基本的な計算手法です. (固有値, 固有ベクトルの計算は大変.)

行基本変形は, 次の 3 つの操作からなります.

1. ある行を (0 でない) 定数倍する.
2. 2 つの行を入れ替える.
3. ある行に別の行の定数倍を加える.

この中で, 2 番目はうまくプログラムを組むことで, 動作の高速化ができます. 行列を単純な 2 次元配列とするのではなく, 行ベクトルへのポインタの配列にするのです. すなわち, 列ベクトルの大きさを持つポインタ配列 `m[]` を準備し,

m[0] → 第 0 行ベクトル
m[1] → 第 1 行ベクトル
...
m[n] → 第 n 行ベクトル

という風にします. こうすると, i 行と j 行の交換は, `m[i]` と `m[j]` の交換で済みます. (素朴に行ベクトルの値の交換は, 行ベクトルの個数だけ実行しなければならない.) また, このようにデータを用意しておくと, i, j 成分は `m[i][j]` でアクセスでき, 行列計算 (和や積) のプログラムも容易です.

3 動的なメモリの確保 (malloc())

行列計算や連立方程式を解く等のプログラムを少し汎用的に書こうとすると, 行列の大きさや連立方程式の本数も変化する量としてプログラムを書く必要が出てきます.

上のように配列を利用するのですが, 可変長配列というのが C11(2011 年制定の規格) で使えるようになりました. しかし, それは局所的な使い方 (ある関数の内部だけでしか使えない) しかできませんので, これから書くようなプログラムには向きません.

C では `malloc()` というライブラリ関数があり, これはシステムに引数で与えらえる量のメモリの確保を要求するものです. 返り値は, 以前解説した `void` 型のポインタで, これを必要な型にキャストして使います. `malloc()` で確保したメモリは, 不要になれば `free()` という関数で, 再び確保可能な領域に戻すことができます. `fopen()` と `fclose()` と似たような関係です. ファイルと同様に, プログラム終了時には, `malloc()` で確保したメモリは自動的に `free()` されます.

`malloc()` は, メモリ確保に失敗すると, NULL ポインタを返します. 以前に述べましたが, NULL ポインタは,「どこも指していない」ことが明確になっているポインタです.

多量にメモリを使うプログラムやネットワークサービスをするプログラムでは, `free()` を呼ぶのは重要ですが, そうでないプログラムでは, `free()` を呼ぶ必要はありません. (この部分は `fclose()` と少し違う.)

上で述べたことを踏まえて, 下の場所にあるプログラムの解説をします.

`https://github.com/okumuralab/algo-c/blob/master/src/matutil.c`

上のソースをダウンロードして, コンパイル, 実行してみてください. main() の前後にある `#if 0` と `#endif` を取り除くかコメントアウトすると, 実行可能ファイルが作れるソースになります.

レポート問題: 締め切り 12 月 20 日 (木)

件名: List 13-10, 送り先は, gengo@math.u-ryukyu.ac.jp

1. list 13–10 において, char 型の配列変数 buf の配列数を 16 より小さい値, 極端な場合 1 にしたプログラムを考える. これについて,
   - コンパイルは可能か?
   - 実行ファイルができた場合, 実行時に何が起こるか?

   を調べてレポートせよ.

   テキストの置き場所: ftp://ftp.math.u-ryukyu.ac.jp/pub/gengo/2018-2/