計算機言語 I 第 13 回
ファイルの読み書き

この資料: http://www.math.u-ryukyu.ac.jp/~suga/gengo/2020-1/13.pdf

レポートへのツッコミ

レポート問題に対する私の解答は, 次です. プログラムをどのように書くかというのば, 難しい問題ですが, 次の工夫をしてあります. 点数部分は, 皆さんと同じように mod 100 を使いました.

- 入学年度, 学年等をマクロ置換で与えたので, この値を書き換えれば, 年度, 学部等が簡単に変えられる.
- チェックディジットは, 条件分岐ではなく, 文字列の場所を利用する.

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

#define YEAR 18   /* 入学年度 */
#define FAC   3   /* 学部番号 */
#define DEPT  1   /* 学科番号 */
#define NUM  40   /* 学生数   */

int main(void)
{
    char checkdigits[] = "BAKJHGFEDC";
    int i, cd;

    for (i=1; i<= NUM; i++){
       cd=((7*(YEAR/10)+6*(YEAR%10)+5*FAC+4*DEPT+3*i/10+2*(i%10))%11)%10;
       printf("%02d%d%d%02d%c %3d\n", YEAR, FAC, DEPT, i, checkdigits[cd], rand()%100);
    }
    return 0;
}
```

プログラムでファイルを読み書きする.

今回は教科書 8 章の解説をします.

ファイルの作成

とりあえず読み込むファイルを作成します. ファイルの中身は次の形です.

```
183101B   7
183102A  49
183103J  73
...
```

みなさんが前回のレポートで提出したプログラムは, 出力で学籍番号の後に改行が入る人がいますが, それは, 改行を入らないようにプログラムを書き直して下さい. あるいは, 最初のページにある私のプログラムを使って下さい. とりあえず, 上で書き換えた前回のレポート (あるいは私が全ページで提示したプログラム) をコンパイルし, その後, リダイレクションでファイルを作ります. 実行可能ファイルの名前は環境毎に違うので, ここでは, UNIX (Linux や macOS) で例示します. (WSL や Visual Studio ではファイル名が違ってくる.)

```
./a.out > input.data
```

プログラムでのファイルの読み書き

ここでは, ファイルシステムにアクセスするためのライブラリ関数を利用する方法を述べます. UNIX(Linux, macOS) では, ライブラリ関数利用以外にも, よりシステムに近い「システムコール利用」でファイルを読み書きする方法もありますが, ここでは述べません. (Windows のシステムコール利用は, 詳しくは知りませんが, とても複雑なようです.)

ライブラリ関数を用いたファイルの読み書きは, C. の処理系では, OS にあまり依存しないものが提供されているのが通常です. ここで述べるのは, その内容です.

Linux (Unix) の C によるプログラミングでは, プログラムとその外部とのデータのやり取りは, 全てファイルの読み書きという形で行うように設計されています. 計算機概論で述べたように, 標準入力, 標準出力, 標準エラー出力というファイルがあり, キーボードは標準入力からの読み込み, 画面への出力は標準出力への書き込みです. ここで述べるのは, これら以外の通常ファイルの利用方法です.

上で用いたリダイレクションで, プログラムとファイルとのやり取りは可能なのですが, それでは, 複数ファイルからの入力などができません. そこで, 通常ファイルを読み書きするためのライブラリ関数が用意されています.

File pointer **を用いた読み書き**.

ライブラリ関数を利用してファイルを読み書きする基本的な方法は, 次です.

1. FILE 構造体へのポインタ変数を宣言する.
2. fopen() 関数で読み書き対象ファイルを開く.
3. fscanf(), fprintf() などでファイルに対する読み書きを実行
4. 読み書きが終われば, fclose() で読み書き対象ファイルを閉じる.

標準入力, 標準出力, 標準エラー出力は, fopen() を利用せずとも開いているファイルなのです.

fopen()

指定されたパスにあるファイルを開き, それへのポインタを返します. 開かれたファイルは「ストリーム (stream)」と呼ばれ, データの流れがその語源だと思われます. ファイルを開くときには, 読み書きのどれするのか, データのどの場所から開くか等の指示をします.

fprintf(), fscanf()

printf(), scanf() に対応するファイルに対する書き込み, 読み込みのためのライフラリ関数です. 標準入出力では, 基本的にテキストデータの読み書きしかしませんが, 通常ファイルだとバイナリデータの読み書きも必要になります. したがって, それらに対応するために, fread(), fwrite() などの関数も用意されています.

上に述べた標準入力, 標準出力は, fprintf(), fscanf() を用いて,

```
fprintf(stdout, ...);
fscanf(stdin, ...);
```

と書いても同じです. stdin, stdout は常に開かれているのです. 同様に, 標準エラー出力への出力も, 次のように記述できます.

```
fprintf(stderr, .... );
```

fclose()

読み書きが終了したファイルは, 関数 `fclose()` で閉じます.

実際には, プログラム終了時点で, プログラムから開かれたファイルは, 自動的に閉じられます. しかし, 次の理由により, `fclose()` をプログラムで使うべきであるとされています.

1. 1 つのプログラムが開くことができるファイルの数に上限がある.
2. `fprintf()` でのファイルの書き込みのエラーへの対処.

1. については, 例えば, 私の手元にある macOS(bash) の環境だと, 1 つのプログラムが開けるファイルの総数は, 256 に設定されているようです (この設定は変更できますが...).

2. に関してですが, ファイルへの書き込みは, 例えば `fprintf()` が呼ばれた際にすぐに起こる事が保証されません. 2 次記憶装置は, CPU に比べて動作が遅いので,「本当の書き込みは, システムが暇なときに実行する」というように作られています. (Buffered I/O). しかし, `fclose()` が呼ばれると, その書き込みを実行します. その時に `fclose()` の返り値で書き込みエラーを検出できます. `fclose()` 自体, その書き込みエラーをリカバーすることはできないのですが, プログラムとして, そのエラーから派生する次のエラーを回避することができるようになります.

実習

さて, 上のことを踏まえて, 教科書にあるプログラム 8.1 を上で作った `input.data` に合わせて書いたのが次です. このプログラムでは, `fscanf()` が読み込みに成功したデータの数を返すことを利用しています. また, エラーは標準エラー出力に出すのが通常です. `exit()` は引数値を shell 変数 `status` に返してプログラムを終了するライブラリ関数です.

```
/* Filename readdata1.c. */

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

int main(void)
{
    FILE *fp;
    char number[10];
    int score;

    if ( (fp = fopen("input.data", "r"))==NULL){
        fprintf(stderr, "input.data を開けません.\n");
        exit (1);
    }

    while (fscanf(fp, "%s%d", number, &score)==2){
        printf("%s %2d\n", number, score);
    }

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

ファイルへの書き出しも同様で, ファイルを開くときに書き込みモードを指定する部分だけが違います. ファイルの読み書きに失敗するのは, ファイルが存在しない以外にも, ファイルやディレクトリの読み書きのモードの設定で, そのファイルやディレクトリの読み書きができない時は失敗します.

ファイルを構造体に読み込む

前回, 構造体の講義をしました. 今回のデータを前回定義した構造体配列に読み込むプログラムが次です. プログラムでは, データ量が 40 人分であることを利用しています.

```
/* FIlename readdata2.c. */

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

typedef struct seiseki {
    char number[10];
    int score;
} seiseki_t;

int main(void)
{
    FILE *fp;
    seiseki_t data[40];
    int i;

    if ( (fp = fopen("input.data", "r"))==NULL){
        fprintf(stderr, "input.data を開けません.\n");
        exit (1);
    }

    for(i=0; i< 40; i++){
        fscanf(fp, "%s%d", data[i].number, &data[i].score);
        printf("%s %d\n", data[i].number, data[i].score);
    }
    fclose(fp);
    return 0;
}
```

上のプログラムでは, seiseki_t 型の 40 個の配列に, input.data の値が読み込まれています. このようにしておけば, さまざ生データ処理を付け加えるのは簡単です.

ファイルへの書き込み

書き込みは, `fopen` で書き込みモードを示す `w` (write) という文字で, ファイルを開き, `fprintf()` でファイルポインタを利用して書いていきます.

出力先が標準出力からファイルに変わっただけですので, 難しくないはずです.

その他, プログラムからファイル読み書きのときの注意

基本的には, 教科書を読んでください. バイナリファイルの利用は,「ファイルサイズを小さくする,」「ファイルの中身を隠す (簡単には何のデータかわからなくする.」などの意味はありますが, 「他のアプリケーションでの利用が難しくなる.」という副作用もあることを注意すべきです.

この講義について

今回でこの講義は終了にします.

単位は, レポートをコンスタントに提出している人 (受講生のほとんどの人) は, 心配しないでください.

後期は, 三柴先生による計算機言語 II が講義されます. 昨年の講義は, 数値計算を中心に取り上げましたが, 今年度はどうなるかは, シラバスをお読み下さい.

4 年次でコンピュータ関係のゼミを希望するなら, 後期の計算機言語 II もしっかり受講しておいて下さい. そうでない限り, (少なくとも私は) 4 年のゼミでコンピュータ関連のことはやりません.

レポート問題: 締め切り 8 月 3 日 (月) AM 10:00 (JST)

`input.data` が試験結果だとして, 次のようなプログラムを書け.

1. 上の最初のプログラムと教科書プログラム 8..2 を参考に, 5 段階絶対評価をファイル `result1.txt` に書き込むプログラムをファイル `result1.txt` の各行は, 学籍番号, 点数, 評価の順に並べる. 5 段階絶対評価は, 80 点以上を A とし, 以下 20 点刻みで, B, C, D, E と評価する.
   件名: gengo2020-1 report 13-1.

2. 2 番目のプログラムを利用して, 5 段階相対評価をファイル `result2.txt` に書き込むプログラムを書け. ファイル `result2.txt` の各行は, 学籍番号, 点数, 評価の順に並べる. 5 段階相対評価は, 平均点を m とすると, $m - 0.5\sigma \leqq \texttt{score} < m + 0.5\sigma$ なら 3 とし, 1σ 毎に評価を上げ下げする. ここで σ は標準偏差である. `result2.txt` の最後の行には, 平均点と標準偏差の値も出力すること.
   件名: gengo2020-1 report 13-2.