Chapter 4

ファイル入出力ならびに構造体

4.1　ファイル入出力

プログラムに対する入力、と聞いてどんな入力のための手段を思い浮かべるだろうか。キーボードからの入力[1] がおそらく真っ先に思い浮かぶだろうと思う。ソフトウェアを作る上でこれと勝るとも劣らずだいじなのは、やはりファイルからのデータの読み込みである[2]。ファイル入力の基本的な構造は次のとおりである[3]。ド定番なので、丸暗記をすべきレベルのものである。なお、ファイルがないだとか権限がないだとかでオープン出来なかったとき(fp==NULLとなる)のためにエラーの処理もしている。

[1]いわゆる標準入力ってやつだ。

[2]これをファイル主体に考えて出力と考える学生がいたりする。通常はプログラム(=ソフトウェア)になったつもりでデータが入ってくるか出ていくかを考えてもらえばまちがいない。そうすると、ファイルからのデータの読み込みは間違いなく入力だ。同様に、ファイルへの書き込みは出力ということになるね。

[3]当然であるが、例えばmain関数でファイル入力を行うのであれば、main関数のなかにこれを書いてやらないとそのままでは動かない。

```
FILE *fp;
char line[1000];
fp=fopen("/home/username/filename", "r");
if(fp==NULL){
     printf("エラー発生");
     return -1;
}
while(fgets(line, sizeof(line), fp)!=NULL){
     /*
          文字列lineを使った処理をここに書く。
     */
}
fclose(fp);
```

同様に、出力と言えば一番身近なのはやはり画面出力[4]であるが、ファイルへのデータの書き込みも同様にだいじだったりする。特に、ファイルに対する書き込みは、既にファイルがある場合に、その上に上書きをするのかそれとも追記するのかでプログラムの持つ機能が変わってくる。例えばスケジュールを記録するソフトウェアでは、予定をファイルに記録するときに既にファイルに記録されている予定データを上書いたりはしないはずだ[5]。つまり、そんな場合は「追記」でないと処理として役に立たないわけだ。fopen関数でファイルをオープンするが、たとえば「5月3日にデート。井の頭公園でボートに乗る。」[6]などというううらやましい予定を書き込むのであれば、プログラムの該当部分は次のようになる。

[4]標準出力ってやつだね。

[5]でなければ、ソフトウェアに覚えておいて欲しい予定データが、予定を新しく追加するたびに消えて行ってしまうことになってしまう。

[6]うーん、フラグ・・・

```
  FILE *fp;
  fp=fopen("/home/username/filename", "a");
  if(fp==NULL){
    printf("エラー発生");
    return -1;
  }
  fprintf(fp, "5月3日にデート。井の頭公園でボートに乗る。
\n");
  fclose(fp);
```

などとすればよいわけだな。fopenでモードを“a”としていることに注意だ。もちろん上書きならこれが“w”になる。この「モード」には読み書き両方OKな“$r+$”や“$w+$”、バイナリモード[7]である“rb”や“wb”などもある。詳細はC言語の教科書にゆずるが、とにかくファイルへの入出力の仕方にはいろいろなニーズがあり、それはソフトウェアの仕様として要求される要件(requirement)[8]によって適切に入出力方式を選ばなければならない。逆に言えば、C言語はそのようなニーズに応えることができるモードを用意しているわけである。モードがいろいろあるのはダテじゃない。

4.2 構造体

直接、入出力に関係するわけではないが、次のような構造のデータをファイルから読み込んでプログラムで使いたいときに構造体はよく使われる。

```
如月, 72, 60, 82 , 65, 80
歌星, 90, 95, 89, 92, 88
城島, 89, 98, 70, 100, 79
朔田, 99, 80, 79, 78, 81
            ⋮
```

各行がカンマ「,」で区切られているこのようなファイルをCSV(Camma Separated Values)と呼ぶ。このCSVの1列目が人名、2列目が英語の成績、3列目が数学の成績、4列目が国語の成績、5列目が理科の成績、6列目が

[7]文字列単位での書き込みではなく、バイト列として書き込みするという意味。
[8]このあたりの話は3年生のソフトウェア工学1や高度情報演習1Bで勉強する。

社会の成績をそれぞれ表しているとしよう。上では都合上、4人しか示していないが、実際には1000人いたとしよう。これをプログラムで扱おうとすると配列を利用すると考えるのはとても自然で、発想としてはとても正しい。では、どのように配列を使うだろうか？

学生諸君をみていると、

- 人名の配列 char name[1000][50]
- 英語の点数の配列 int english[1000]
- 数学の点数の配列 int math[1000]
- 国語の点数の配列 int japanese[1000]
- 理科の点数の配列 int science[1000]
- 社会の点数の配列 int socio[1000]

などとすることが多かろうと思う[9]。だが、name[0](如月君)の英語の点数がenglish[0]であるというのは、プログラマが決めた約束事でしかなく、仕様上使いにくいデータの持ち方となってしまう場合が少なくない。

例えば、数学が得意な順に学生の情報を並べる（ソートする）必要がある場合、配列math[ ]の要素を並び替えるだけでなく他の配列の要素もmath[ ]の並びに合わせて並び替えなくてはならない。

では、どのようにすると便利だろうか。次の構造体を考えてみよう。

```
struct _student{
    char name[50];
    int english;
    int math;
    int japanese;
    int science;
    int socio;
};
typedef struct _student STUDENT;
STUDENT score[1000];
```

struct _studentは、“_student”という名前の構造体（=変数のセット)を作れということを表しており、 name、english、math、japanese、science、

[9]実際、プログラムが苦手な学生はこのようにする傾向があるようである。

socioという変数を内部にもっている[10]。名前と英数国理社の各成績が、ひとりの学生の情報のひとまとまりだと考えればそれほどむずかしくないとおもう。 score[1000]は、このセットが1000個あるということを表している。この構造体の配列がもつ情報は、前の6つの配列がもつものと同じではある。しかし、数学の点数が高い順に並べるとすると、次のようなコードですむ[11]。

```
int i,j;
STUDENT s;
for(i=0; i<1000; i++){
   for(j=i+1; j<1000; j++){
      if(score[i].math < score[j].math){
          s=score[i];
          score[i]=score[j];
          score[j]=s;
      }
   }
}
```

これを6つの配列を使って同じことをやろうとすると

```
          s=score[i];
          score[i]=score[j];
          score[j]=s;
```

と3行で済む部分を、各配列の値の入れ替えとして実装しなければならないから、6つの配列×3行= 18行を使って実装することになる[12]。この例では、CSVが6列だからまだよかった。現実には数十列あるCSVを扱うこともザラであるから、ソフトウェアの実装をするときには、こんなところがバグの温床になりやすい [13]。構造体は、最初とっつきにくいかもしれないが、プログラムを簡潔にするためにはとてもだいじなのである。それに、これを使いこなすと

---

[10]typedefの行は、型宣言をするときにstruct _studentと毎回書くのが大変なので、型の別名をSTUDENTとしていることを意味している。

[11]「データ構造とアルゴリズム」でもっと効率的なアルゴリズムを学ぶはず。

[12]さらに、nameなどは文字列なので値の入れ替えはint型などに比べたら手間である。なおさら、いや～な感じである。

[13]ある列の値を入れ替え忘れた！とかね。プログラムを作るより、バグを探すほうがはるかに大変な作業になるから、これは結構つらい。

オブジェクト指向[14]

にとても入りやすいというおまけ付きだ。マスターしない手はない。

[14]詳しくは上級プログラミングで。

では、演習に移ろう。授業時間中に教員・TAが諸君の席をまわるので疑問点はこの場で解決すること。また、次回（第4回）までに、セクション4.4まで完成させ、セクション4.3〜4.4のソースコード、出力をレポートとしてまとめ提出せよ。次回の授業開始時にレポートを集めるので、開始までにはレポートを仕上げておき、提出できるようにしておくこと。セクション4.5の発展問題に関しては、レポートに含めると回答問題数と出来により加点する。是非チャレンジすること。

4.3 ファイル入出力

- 先回作ったプログラムのソースファイル“prog2-1a.c”をカレントディレクトリにコピーせよ。
- ファイルprog2-1a.cの各行を一行目から画面に出力するプログラムを作成し、実行せよ。(prog3-1a.c)　コンパイル後の実行プログラムはprog3-1aとすること。
- 下記のようにファイルprog2-1a.cの各行に行番号を付け画面に出力するプログラムを作成し、実行せよ。(prog3-1b.c)　コンパイル後の実行プログラムはprog3-1bとすること。

```
1:#include <stdio.h>
2:
3:int main(int argc, char **argv){
4:  int i,j;
```

⋮

- ファイルprog2-1a.cの各行に行番号を付けたものをファイルprog2-1a-line.txtに書き出すプログラムを作成し、実行せよ。(prog3-1c.c)　コンパイル後の実行プログラムはprog3-1cとすること。

4.4 CSVファイルを読み込み、構造体を使ってデータを操作してみる

- population.csvは、各都道府県の人口データである。一列目は都道府県名（以下、県名)、二列目はその県の人口を表している。なお、

プログラムを簡単にするためファイルの各行はカンマ区切りではなく半角スペース区切りとしている。

- 以下の構造体を使うこと。ただし、nameは県名を表し、populationは人口を表しているものとする。

```
struct _pref{
    char[] name;
    int population;
};
typedef struct _pref PREF;
```

- 各県のデータを要素に格納するため、PREF型の要素を持つ配列を用意せよ。population.csvを先頭から1行ずつfscanf関数を用いて読み込み、構造体の要素に格納する。なお、文字列をint型に変換するにはatoi関数を用いよ[15]。そのうえで、各県の人口の平均値と分散をそれぞれ求め、画面に表示するプログラムを作成し、実行せよ。(prog3-2a.c)コンパイル後の実行プログラムはprog3-2aとすること。

- prog3-2a.cを改良し、各県のデータを人口が少ない順に並べ、県名と人口を少ないほうから20件、画面に出力するプログラムを作成し、実行せよ。(prog3-2b.c)コンパイル後の実行プログラムはprog3-2bとすること。

4.5 発展問題

- 4.3のプログラムを応用し、コマンド“prog3-3a aaa.txt bbb.txt”を実行すると、aaa.txtの各行に行番号がついたファイルがbbb.txtにコピーされるようなプログラムを作成・実行せよ。(prog3-3a.c)コンパイル後の実行プログラムはprog3-3aとすること。

[15]string.hをインクルードすることも忘れずに。