

ファイル入出力

山本昌志*

2004年11月09日

1 前回の復習と今週の内容

1.1 前回の復習

配列を学習するまで、諸君が知っていたデータ構造は、単純型と呼ばれるもので、

```
int a;  
double x;
```

のように宣言する。これは、

- a という名前が付いた整数を格納する入れ物を用意する。
- x という名前が付いた倍精度実数を格納する入れ物を用意する。

というように解釈する。このデータ構造では、変数名、たとえば a や w を指定することで、データにアクセスできる。

先週は、単純型に比べ、より多くのデータを扱える配列というデータ構造について、学習した。それは、

```
int b[10000];  
double y[10000];
```

のように宣言する。配列名と自然数により目的のデータにアクセスできる。この配列を上手に使うことにより、大量のデータを取り扱うことができる。このようなことを学習した。

1.2 今週の学習内容

前回までの学習で、配列を使うことにより、大きなデータを扱えるようになったが、依然としてデータの入力は、キーボードを用いていた。また、データの表示もディスプレイのみであった。これでは、実際には大きなデータを扱うことは難しい。そこで、本日は、データをハードディスク上で読み書きする方法を学習する。

*国立秋田工業高等専門学校 電気情報工学科

2 ファイルの種類

2.1 ファイルと C 言語プログラム

1つの単位として取り扱えるデータの集合をファイルと言い、その集合には名前(ファイル名)が付けられている。通常は、ハードディスクのような外部記憶装置上に保存されている。ファイルはオープンすることによって、ストリーム¹と関連付けられる。

Cプログラムがファイルをオープンすると、ファイルとCプログラムは、ストリームによって結ばれる。これを、図??に示す。Cプログラムとファイルを結びつけるものがストリームというものである。このストリームと言う概念は、他のプログラミング言語でも出てくるので覚えておいた方が良い。

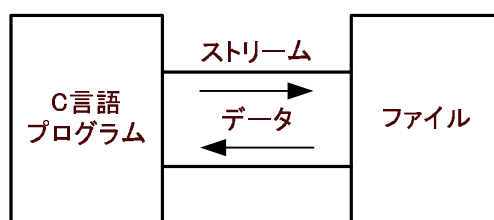


図 1: プログラムとストリーム、ファイルの関係

ここでは、ほとんどの場合、ファイルはハードディスクを示すが、実際にはディスプレイやキーボード、メモリーなど様々なデバイスである。

2.2 バイナリーファイルとテキストファイル

ストリームに流れるデータは、バイナリーデータとテキストデータがある。バイナリーデータ²の場合、コンピューターで取り扱っているデータそのものである。一方、テキストデータの場合、データは ASCII 文字に変換される。これでは、何のことも分からないと思うので、データ(整数の 1234 と文字列"akita")をテキストとバイナリーで保管されたファイルを、図 2 と 3 に示す。テキストデータの方は、通常のテキストエディターで内容を見ることができるが、バイナリーデータの方は意味不明の文字が書かれる。バイナリーデータを見るためには、図 5 のようにバイナリーエディターで見るしかない。それでも、書かれている内容を理解することは難しい³。

ようするに、テキストデータの方が簡単である。この授業ではテキストデータしか取り扱わないことにする。

¹stream:流れ

²binary:二進数の

³ $(1234)_{10} = (4d2)_{16}$ で、リトルエンディアンで記録している

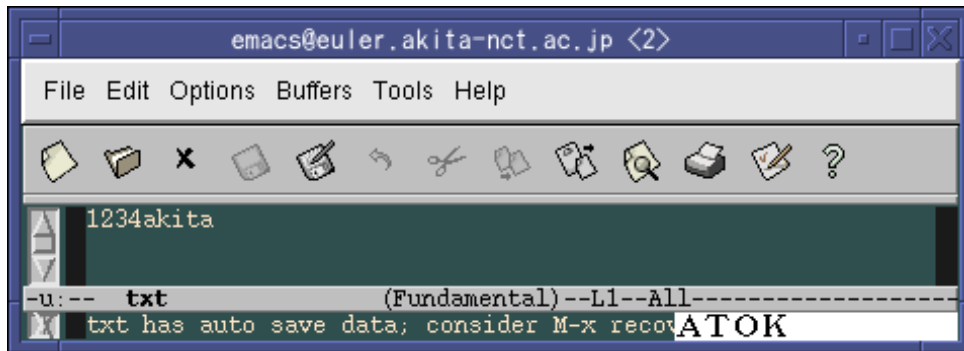


図 2: テキストデータ。整数の 1234 と文字列”akita”。

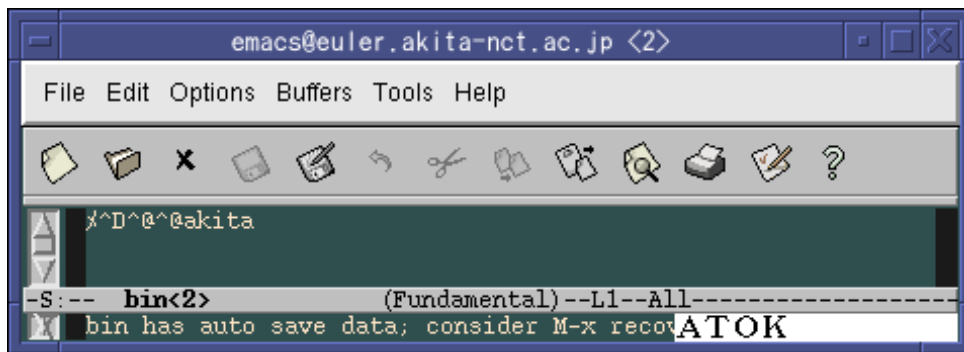


図 3: バイナリーデータ。整数の 1234 と文字列”akita”。

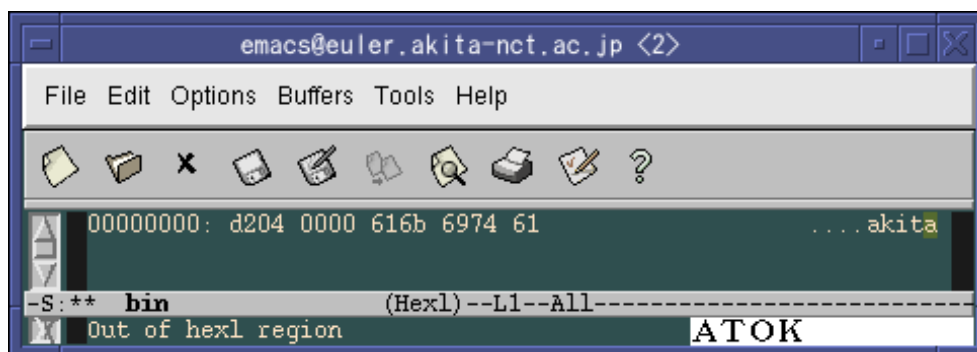


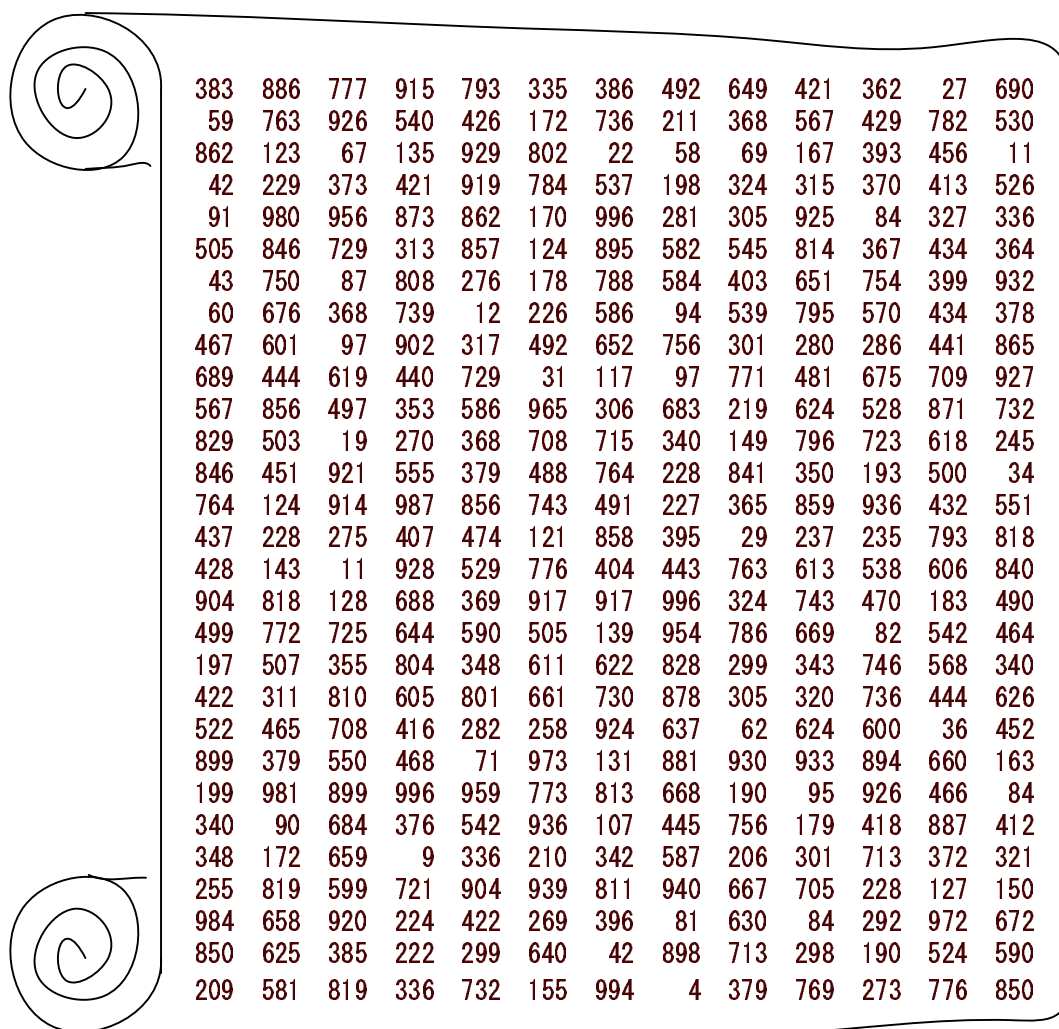
図 4: バイナリーデータをバイナリーモードで表示。

2.3 シーケンシャルアクセスとランダムアクセス

ハードディスク上に書かれたデータは、長い巻物に文字が書かれていると思えばよい。巻物であるから、ページ概念はないが、行はある。また、1行に書かれる文字数の制限も無いと考えてよい。

この巻物に大量の文字が書かれているとしよう。この文字を、巻物の始めから、そして左から順番に読むのをシーケンシャルアクセスと言う。一方、あっちこっちに飛んで、いろいろな場所をつまみ食いのように読むのをランダムアクセスと言う。これらの違いを理解しておくことが大事である。

この授業では、取り扱いが簡単なシーケンシャルアクセスのみを学習する。ランダムアクセスについては、興味のある者は、自分で学習せよ。



383	886	777	915	793	335	386	492	649	421	362	27	690
59	763	926	540	426	172	736	211	368	567	429	782	530
862	123	67	135	929	802	22	58	69	167	393	456	11
42	229	373	421	919	784	537	198	324	315	370	413	526
91	980	956	873	862	170	996	281	305	925	84	327	336
505	846	729	313	857	124	895	582	545	814	367	434	364
43	750	87	808	276	178	788	584	403	651	754	399	932
60	676	368	739	12	226	586	94	539	795	570	434	378
467	601	97	902	317	492	652	756	301	280	286	441	865
689	444	619	440	729	31	117	97	771	481	675	709	927
567	856	497	353	586	965	306	683	219	624	528	871	732
829	503	19	270	368	708	715	340	149	796	723	618	245
846	451	921	555	379	488	764	228	841	350	193	500	34
764	124	914	987	856	743	491	227	365	859	936	432	551
437	228	275	407	474	121	858	395	29	237	235	793	818
428	143	11	928	529	776	404	443	763	613	538	606	840
904	818	128	688	369	917	917	996	324	743	470	183	490
499	772	725	644	590	505	139	954	786	669	82	542	464
197	507	355	804	348	611	622	828	299	343	746	568	340
422	311	810	605	801	661	730	878	305	320	736	444	626
522	465	708	416	282	258	924	637	62	624	600	36	452
899	379	550	468	71	973	131	881	930	933	894	660	163
199	981	899	996	959	773	813	668	190	95	926	466	84
340	90	684	376	542	936	107	445	756	179	418	887	412
348	172	659	9	336	210	342	587	206	301	713	372	321
255	819	599	721	904	939	811	940	667	705	228	127	150
984	658	920	224	422	269	396	81	630	84	292	972	672
850	625	385	222	299	640	42	898	713	298	190	524	590
209	581	819	336	732	155	994	4	379	769	273	776	850

図 5: ファイルのイメージ。

3 ファイル入出力の例

3.1 ファイル出力

細かい文法の話をする前に、実際のファイル出力の例を示す。以下のプログラムがファイル出力の例である。このプログラムを実行すると、"output.txt"というテキストファイルができる。このテキストファイルは、図6のようになっている。

このプログラムの中で、ファイル処理に関する部分は、次の通りである。

- 「FILE *fp」でファイルポインター fp を宣言している。ファイルポインターとはファイルの情報を入れておく変数だと思ってほしい⁴。
- 「fp = fopen("output.txt", "w")」で"output.txt"と言うファイルを、書き込みモード ("w") で開いている。そのファイルの情報は、fp に代入される。
- 「fprintf(fp, "%3d\t", 10*i+j)」で、ファイルに書き込んでいる。ファイルポインターを示す引数以外は、printf 文と同じである。
- 「fclose(fp)」でファイルを閉じている。

実際のファイルの書き込みは、printf 文とほとんど同じなので、簡単である。ただ、前後にファイルのオープンとクローズがあるだけである。

```
#include <stdio.h>

int main(void)
{
    int i, j;
    FILE *fp;

    fp = fopen("output.txt", "w");

    for(i=0; i<=99; i++){
        for(j=0; j<=9; j++){
            fprintf(fp, "%3d\t", 10*i+j);
        }
        fprintf(fp, "\n");
    }

    fclose(fp);

    return 0;
}
```

⁴実際は、ポインターなので変数を入れる場所を示すものである。ポインターについては学習指定なので、この様に書くと覚える。

```

0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169
170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189
190 191 192 193 194 195 196 197 198 199
200 201 202 203 204 205 206 207 208 209

930 931 932 933 934 935 936 937 938 939
940 941 942 943 944 945 946 947 948 949
950 951 952 953 954 955 956 957 958 959
960 961 962 963 964 965 966 967 968 969
970 971 972 973 974 975 976 977 978 979
980 981 982 983 984 985 986 987 988 989
990 991 992 993 994 995 996 997 998 999

```

図 6: 作成されたテキストファイル。

3.2 ファイル入力

ファイル入力は、次のように書く。このプログラムを実行すると、先ほどファイル出力で作成されたファイルのデータを読み込み、配列 `data` に数値が代入される。

このプログラムの中で、ファイル処理に関する部分は、次の通りである。

- 「`FILE *fp`」でファイルポインター `fp` を宣言している。
- 「`fp = fopen("output.txt", "r")`」で"output.txt"と言うファイルを、読み込みモード ("r") で開いている。そのファイルの情報は、`fp` に代入される。
- 「`fscanf(fp, "%d", &data[i][j])`」で、配列に代入している。ファイルポインターを示す引数以外は、`scanf` 文と同じである。
- 「`fclose(fp)`」でファイルを閉じている。

実際のファイルの読み込みは、scanf 文とほとんど同じなので、ファイル出力同様に簡単である。

```
#include <stdio.h>

int main(void)
{
    int i, j;
    int data[100][10];
    FILE *fp;

    fp = fopen("output.txt", "r");

    for(i=0; i<=99; i++){
        for(j=0; j<=9; j++){
            fscanf(fp, "%d", &data[i][j]);
        }
    }

    fclose(fp);

    return 0;
}
```

4 C 言語でのファイル入出力

4.1 ファイル入出力の流れ

どのような言語でもファイル処理の流れは似ている。まず初めにファイルを開いて (open)、その後処理を行い、最後に閉じる (close) のである。その流れを図 7 に示す。

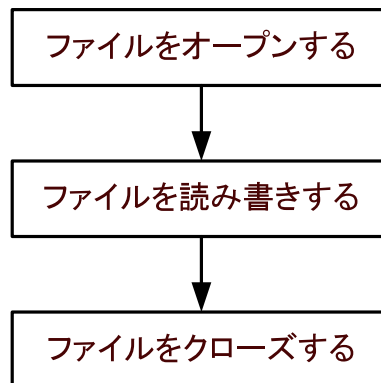


図 7: ファイル処理の流れ。

4.2 ファイルポインター

ストリームを制御するために必要な情報が記述されている。これは変数の宣言と同じで、次のように書く。とりあえず、ファイルを使うときに必要な変数と思ってほしい。

```
FILE *ファイルポインター名;
```

注意すべきことは、

- 変数の型名は、すべて大文字で「FILE」と書く。
- ファイルポインター名の前に、アスタリスク「*」をつける。

である。

4.3 オープンとクローズ

ファイルを使うときには、最初にそれをオープンする必要がある。

```
ファイルポインター名 = fopen("ファイル名", "モード");
```

ファイル名は、絶対パスや相対パスで記述することができる。また、ただ単にファイル名のみがかかれた場合、その実行ファイルがあるディレクトリーとなる。モードは、バイナリーとテキストがある。テキストの場合について、表 1 に示す。いろいろなモードがあるが、とりあえずは、読み込み (r) と書き込み (w) を理解しておけばよい。

表 1: テキストモードのオープン

モード	処理	ファイルがないとき	ファイルがあるとき
r	読み込み (read)	NULL を返す	正常処理
w	書き込み (write)	新規作成	前のファイルは破棄
a	追加書き込み (append)	新規作成	前の内容の後に追加
r+	書き込み (更新)	NULL を返す	正常処理
w+	読み込み (更新)	新規作成	前の内容は破棄
a+	追加読み書き	新規作成	前の内容の後に追加

使い終わったならば、クローズしなくてはならない。

```
fclose(ファイルポインター名);
```

4.4 ファイル出力

ファイル出力は、printf 文とほとんど同じである。

```
fprintf(ファイルポインター, 出力書式, 引数並び);
```

4.5 ファイル入力

ファイル入力は、scanf 文とほとんど同じである。

```
fscanf(ファイルポインター, 入力書式, &引数並び);
```

4.6 特別なファイル

UNIX には、ファイルポインターの指定やオープン、クローズ処理なしで使えるファイルがある。それを、表 2 に示す。「fscanf」や「fprintf」のファイルポインター名を書くところに、これらを書くと、キーボード入力やディスプレイ出力になる。上手に使えば、便利である。

表 2: 特殊なファイル

ファイル	ファイルポインター	デバイス
標準入力	stdin	キーボード
標準出力	stdout	ディスプレイ
標準エラー出力	stderr	ディスプレイ

5 予習

5.1 プログラム作成

以下のプログラムを次回の授業で作成する。次回の授業まで、プログラムの内容を考えてくること。今回は課題としないが、次回の授業で課題を与える。

- 11月の毎日の1時間毎の気温のデータファイルがある。表3のようになっている。
 - 各行には、その日の1時間毎の24個のデータがある。0時～23時までである。
 - 行数は30行で、11月1日から11月30日を表している。
- 日毎の最高気温と最低気温、平均気温をディスプレイに書き出す。
- 11月の最高気温と最低気温、平均気温をディスプレイに書き出す。

表 3: 11月の気温

8.3	7.9	7.5	7.2	...	9.8
9.3	9.2	9.1	9.0	...	6.3
6.2	5.8	5.3	4.9	...	12.0
⋮	⋮	⋮	⋮	⋮	⋮
4.3	3.9	3.3	2.8	...	3.8