目次

★2　復習—2 次元配列

★ 2.1　2 次元配列の宣言と使用

`int a[2][3];` と宣言すると，右図のような 2 次元配列を作ることができる (☆ 1)．添字の範囲は 1 次元配列と同様で，この例では，前の添字は 0 から $1(=2-1)$ まで，後の添字は 0 から $2(=3-1)$ までの整数値をとれる．図では縦横に要素が並んでいるが，メモリ中でこのように並ぶわけではない (☆ 2)．

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]

☆ 1) `int a[5][3][4];` と宣言すれば 3 次元配列を作れるし，同様にしてより高い次元の多次元配列を作ることもできる．

☆ 2) 後述の「メモリ中での 2 次元配列の要素の並び方」参照

Q1. 以下のプログラム `2darray01.c` を実行するとどのような出力が得られるか．また，6 行目を `int m = 2, n = 4;` に変えるとどうなるか．

2darray01.c

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int x[10][10];
 6    int m = 3, n = 2;
 7    int i, j;
 8
 9    for(i = 0; i < m; i++){
10      for(j = 0; j < n; j++){
11        x[i][j] = 100*i + j;
12      }
13    }
14    for(i = 0; i < m; i++){
15      for(j = 0; j < n; j++){
16        printf(" %d", x[i][j]);
17      }
18      printf("\n");
19    }
20    return 0;
21  }
```

★ 2.2 2 次元配列の引数としての受け渡し

1 次元配列を関数の引数として受け渡す場合の典型的な書き方は，次のようなものであった（前回資料参照）.

● 呼び出し側

```
int a[10], n = 7;
      :
f(a, n);
```

● 関数 f の定義

```
void f(int x[], int n) {
   ...
}
```

引数の宣言では，配列の大きさを書かずに済ませることができている．しかし，次の例に示すように，2 次元配列の場合は勝手が違う．下記左のプログラムを試しに関数を用いるように書き換えてみると，右のようになる.

Q2. 2darray02.c を実行して次のように入力すると，どんな出力が得られるか.

```
2 3
1 2 3 4 5 6
```

2darray02.c

```
 1  #include <stdio.h>
 2
 3  int main(void)
 4  {
 5    int x[5][10];
 6    int m, n, i, j;
 7
 8    /* 行数列数の読み込み */
 9    scanf("%d %d", &m, &n);
10    /* 各要素の読み込み */
11    for(i = 0; i < m; i++){
12      for(j = 0; j < n; j++){
13        scanf("%d", &x[i][j]);
14      }
15    }
16    /* 各要素の値を 3 倍 */
17    for(i = 0; i < m; i++){
18      for(j = 0; j < n; j++){
19        x[i][j] *= 3;
20      }
21    }
22    /* 出力 */
23    for(i = 0; i < m; i++){
24      for(j = 0; j < n; j++){
25        printf(" %d", x[i][j]);
26      }
27      printf("\n");
28    }
29    return 0;
30  }
```

2darray03.c

```
 1  #include <stdio.h>
 2
 3  「ここにはプロトタイプ宣言を書く」
      (関数定義の頭と同じものを書いて後ろにセミコロン ‘;’)
 4
 5  int main(void)
 6  {
 7    int x[5][10];
 8    int m, n, i, j;
 :
 :    (2darray02.c の 8-15 行目と同じ)
 :
18    /* 関数 f の呼び出し */
19    f(x, m, n);

20    /* 出力 */
 :    (2darray02.c の 23-28 行目と同じ)
28  }
29
30

31

32

33

34

35

36

37

38
```

2darray03.c の 29 行目の引数の宣言では，2 次元配列の後ろ側の大きさを書かずに済ませることはできない (☆ 3)．int a[][] と書くと，コンパイルエラーになる．また，その大きさは，呼び出し側で宣言した配列の大きさ（上記の例では 10）と一致しなければならない．

☆ 3) ［発展］ その理由は授業では説明しない．1 次元配列の場合は，先頭要素の番地 (a[0] の番地) がわかれば i 番目の要素 (a[i]) がどこにあるかわかるのに対して，2 次元配列の場合，先頭要素の番地と 2 つの添字 i,j に加えて各列の大きさを知らないと a[i][j] がどこにあるか確定できないからである（下図参照）．

★ 2.3 #define のこと

配列の大きさなどの定数は，#define を用いて定義しておくと便利である．array03.c の場合なら，例えば 2 行目あたりに

```
#define NROW 5
#define NCOL 10
```

などと書き，それを利用して，7 行目の変数宣言は int x[NROW][NCOL];，関数定義の先頭（プロトタイプ宣言も）は void f(int a[][NCOL], int m, int n) とすればよい．こうすれば，配列の大きさを変えたくなったときには#define の所だけ変更すれば済み，バグを生みにくくなる．

★ 2.4 メモリ中での 2 次元配列の要素の並び方

C 言語では，例えば int x[2][3] のように 2 次元配列を宣言すると，メモリ中での配列の各要素のならび順は左下の図のようになる．

x[0][0]
x[0][1]
x[0][2]
x[1][0]
x[1][1]
x[1][2]

例 1

```
for(j = 0; j < 3; j++){
   for(i = 0; i < 2; i++){
      x[i][j] を使った計算
   }
}
```

例 2

```
for(i = 0; i < 2; i++){
   for(j = 0; j < 3; j++){
      x[i][j] を使った計算
   }
}
```

メモリアクセスはこの順番通りの方が効率がよいので (☆ 4)，ソース中ではなるべく後ろの添字が先に変化するように（例 2 のように，後ろの添字に関する繰り返しが二重ループの内側にくるように） した方が速く計算できることが多い．

☆ 4) その理由は「計算機システム II」の授業で解説されるかも

★3 リダイレクション

★3.1 標準出力のリダイレクション

2darray01.c をコンパイルして 2darray01 という実行形式のファイルを作ったとする．このとき，次のようにプログラムを実行すると，出力を端末画面に表示するかわりにファイル hoge に書き込むことができる (☆ 5)(☆ 6)．

```
$ ./2darray01 > hoge
```

ファイル hoge は C 言語のソースなどと同じ**テキストファイル** (☆ 7) なので，書き込まれた内容を確認するには，

- cat コマンドを用いる　　$ cat hoge
- less コマンドを用いる (☆ 8)　　$ less hoge
- テキストエディタ emacs を用いる　　$ emacs hoge &

などの方法がある．

上記のように'>' をつけて実行すると，printf() などの出力が，普段の出力先（**標準出力**と呼ぶ）から「向き」を変えられて，ファイル hoge に向けられる．このような操作のことを「**リダイレクト** (redirect) する」あるいは「**リダイレクション** (redirection)」と呼ぶ．より詳しくは「標準出力をリダイレクトする」という．

☆ 5) 入力はキーボードから通常通り受け付けられる．また，ファイル hoge は存在していなければ新たに作られる．存在していれば上書きされる．

☆ 6) 先頭の$は，端末プログラム（コンソール）上で人間とやりとりをする「シェル」というプログラムが人間に文字入力を促すために出す文字．「**プロンプト**」という．

☆ 7) テキストファイル: 文字以外の情報を含まない単純な形式のファイル．ワードプロセッサや表計算ソフトのような専用のソフトウェアを用いなくともテキストエディタ等で内容を確認したり編集したりできる．

☆ 8) less の使い方: カーソルキーや'j','k' でスクロール．終了は，アルファベットの'q'．テキストファイルを眺めるだけ（編集しない）なら，わざわざエディタを立ち上げるより less の方が便利でしょう．

★3.2 標準入力のリダイレクション

今度は入力の方をリダイレクトしてみよう．2darray02.c をコンパイルして 2darray02 という実行形式のファイルを作ったとする．また，同じディレクトリ内に右のような内容のテキストファイル fuga があったとする．

```
3 2
1 2
3 4
5 6
```

このとき，次のようにプログラムを実行すると，scanf() への入力をキーボードから与えるかわりにファイル fuga から読み込ませることができる (☆ 9)．

```
$ ./2darray02  <  fuga
```

上記のように'<'(☆ 10) をつけて実行すると，このプログラムの scanf() は，キーボードからの入力（**標準入力**と呼ぶ）のかわりに，fuga の中身を受け取ることになる．これもリダイレクションである．より詳しくは「標準入力のリダイレクション」という．

'>' と'<' は同時に使うこともできる．次のようにすれば，2darray02 は fuga を入力として読み込み，出力は hoge に書き込むことになる (☆ 11)(☆ 12)．

```
$ ./2darray02  <  fuga  >  hoge
```

☆ 9) 出力は通常通り端末画面に出てくる

☆ 10) 「大なり」じゃなくて「小なり」なのに注意

☆ 11) したがって，端末画面には何も出てこない

☆ 12) $ ./2darray02 > hoge < fuga でもよい

★4 画像を処理しよう

★4.1 画像いろいろ

一般に，コンピュータで扱うデジタル画像は，画素と呼ばれる点が規則的にならんだものである．個々の画素は，色のついていない**グレイスケール画像** (☆ 13) の場合は明るさを表す一つの数値で，カラー画像の場合は色を表す数値の組 (多くの場合，R(赤)，G(緑),B(青) の三つ) で表現される．例えば，現在一般的なコンピュータ用ディスプレイは，横 1280 × 縦 1024 = 約 130 万個 以上の画素を表示可能で，個々の画素は RGB それぞれ 256 段階 (8bit) すなわち約 1678 万色 (☆ 14) を表現することができる．

画像はこのようにたくさんの数字の集まりであるため，大きなデータ容量を必要とする．例えば上記の例では，1 画素あたり 24bit = 3B として，1 画面分の画素の値を記憶するため数 MiB もの容量が必要となる (☆ 15) ．

この調子で画像を記憶していては記憶容量があっという間に足りなくなるので，画像をファイルに保存する際には，通常はデータ容量を減らす工夫（圧縮）をする．世の中には，この圧縮の方法やファイルの中に数字をならべるやり方の異なる**画像形式 (フォーマット)** が数多く存在しており，用途に応じて様々に使い分けられている (☆ 16)．この科目は画像処理に関するものではないので，これらの画像形式の詳細は省略する．興味のある人は，画像処理関係の本を読んだりウェブで検索したりしてみるとよいでせう．

龍谷大学の計算機室の Linux 環境では，ImageMagick や Netpbm という画像処理ツール群を利用できる．例えば，ImageMagick の `display` コマンドを使えば種々の画像を表示することができる．カレントディレクトリの `hoge.jpg` という名前の画像ファイルを表示したければ，次のようにすればよい (☆ 17)．

```
$ display hoge.jpg &
```

☆ 13) グレイスケール画像は，明るさが複数に段階分けされる．白と黒の 2 色しかない画像は **2 値画像**という．

☆ 14) $2^8 = 256$ より $256 \times 256 \times 256 =$ 約 1678 万

☆ 15) $1280 \times 1024 \times 3\mathrm{B} = 3840\mathrm{kB} = 3.75\mathrm{MiB}$

☆ 16) 例えば，JPEG, GIF, PNG, PGM,.... ファイルの拡張子で区別がつくようにしていることが多い．`.jpg`，`.gif`，`.png`，`.pgm` など など．

☆ 17) `display` コマンドを起動して表示されたウィンドウ上にマウスカーソルがある状態で右クリック／左クリックすると様々な機能を呼び出すことができる．終了もそこから．

★4.2 Portable Gray Map (PGM)

この授業では，最もシンプルな画像形式の一つである **Portable Gray Map (PGM) フォーマット**を取り上げ (☆ 18)，この形式の画像を扱う方法を考えていく．名前から推測できるように，これはグレイスケール画像の形式である．PGM には raw/plain の 2 つの形式があるが，ここでは plain 形式のものを扱う．これは非常に単純な構造のテキストファイルであるため，簡単な C 言語プログラムでそれを読み書きしたりいじったりすることができる (☆ 19)．

● plain PGM 形式の画像ファイルのサンプル (☆ 20)

```
P2
640 480
255
35 77 100 9 ... 213
186 123 ...
:
:
```

☆ 18) PGM の仲間に，カラー画像の形式 PPM や 2 値画像の形式 PBM というものもあり，PNM(Portable aNy Map) と総称される．

☆ 19) その分ファイルサイズが非常に（本来のデータ量よりも）大きくなるので，実用的ではない．

☆ 20) 本当はヘッダ部 (P2 や幅，高さを記した部分）に`#`ではじまるコメント行を挿入することができる．また，画素値のならびは 1 行 70 文字以内となるように改行するきまりである．この授業では，かんたんのため，これらのきまりを無視している．

Q3. 以下の PGM 画像の幅と高さはそれぞれ何画素か．どのような見た目か（改行位置が画像の幅と対応しているとは限らないことに注意）．

```
P2
200 100
255
0 1 2 3 4 5 ... 98 99
0 1 2 3 4 5 ... 98 99
        :
（全部で 200 行繰り返し）
        :
0 1 2 3 4 5 ... 98 99
```

plain PGM 形式の画像を処理するプログラムは，以下のように，これまで学んできたことを応用して作れる (☆ 21)．

1. 画像を入力: `scanf()` と標準入力のリダイレクションを使えば，PGM 画像のファイルから画素値を読み込めそう
2. 画像をいじる: 2 次元配列に画素値を格納して，それをいじればよさそう
3. 画像を出力: `printf()` と標準出力のリダイレクションを使えば，PGM 画像のファイルを作れそう

☆ 21) ここでは `printf` や `scanf` を使う例で説明しているが，もちろん他の文字・文字列入出力関数を用いても構わない．