3　配列

1 つのデータの値を記憶したり操作したりするのがオブジェクトである。複数の同じ型のデータをまとめて記憶したり操作したりするのが**配列** (array) である。配列を表す配列型は同じ要素型【型】のオブジェクトを一定個数連続的に割り付けたもので、同じ型の多くのデータを扱うとき用いる。

3.1　1 次元配列

3.1.1　配列宣言

配列を用いるときは、オブジェクトと同様に宣言が必要である。

```
要素型 識別子 [式];
```

識別子 [式] は**配列宣言子** (array declarator) である。式は整数型で、配列がもつ要素数になる。式が定数式の場合は正の整数でである。

例 3.1 宣言を、

```
int a[3];
```

とすると、要素型は `int`、配列宣言子は `a[3]` である。識別子は `a` である。配列 a は int 型の配列で 3 個の `int` 型オブジェクト `a[0],a[1],a[2]` を持つ。`a[0],a[1],a[2]` は配列 `a` の要素 (element)、0,1,2 は配列 `a` の添字 (subscription) と呼ばれる。角括弧 `[]` は添字演算子 (subscript operator) である。

演習 3.1 宣言が、

```
char c[20];
double d[30];
```

となっているとき、配列 `c`、`d` について分かることを説明せよ。

メモリは 1 バイト (8 ビット) 毎に番地 (アドレス) がつけられている。番地は 10 進数または 16 進数で表されるが、本講義では 16 進数で表す。

配列はメモリに連続して確保される。(東女の)`int` 型は 4 バイト (32 ビット) であるので、仮に `a[0]` が 0x10000 番地 (0x は 16 進数を表す。) に格納されたときは、

アドレス	要素名	内容
0x10000	`a[0]`	
0x10004	`a[1]`	
0x10008	`a[2]`	

となる。

注 3.1 0x1000C 番地には別のデータが入っているが、間違えて `a[3]` に値を代入すると、0x1000C 番地のデータが強制的に書き換えられ、場合によってはプログラムが暴走することがある。

同じ要素型の複数の配列を宣言するときは、識別子 [式] をカンマ',' で区切って

要素型 識別子 [式], 識別子 [式], 識別子 [式];

のように並べる。

例えば、

```
int a[3],b[4];
```

とすると、7 個の int 型データ a[0],a[1],a[2],b[0],b[1],b[2],b[3] が利用できる。

3.1.2　配列の操作

代入や演算はオブジェクトと同様に扱える。

```
a[0] = 12;
a[1] = 34;
b[0] = a[1] - a[0];
```

とすると、a[0],a[1] に int 型定数 12、34 が代入され、b[0] に a[1]-a[0] すなわち 22 が代入される。

添字には int 型のオブジェクトを用いることができ、添字を計算することもできる。

```
i = 0;
a[i] = 12;
a[i+1] = 34;
b[i] = a[i+1] - a[i];
```

としても上と同じ結果となる。

a[0],a[1],a[2] を b[0],b[1],b[2] にコピーするときは、

```
for(i = 0;i  <  3;i++)
  b[i] = a[i];
```

とする。たとえ配列の要素型と大きさが等しくても

```
b = a; /* 正しくない */
```

とはできない。

3.1.3　配列の初期化

1 次元配列宣言の初期化の構文規則は、

要素型 識別子 [式] = { 初期化子並び };

である。{ 初期化子並び } を**初期化子**という。初期化子並びは、カンマ',' で区切った式 (定数または文字列リテラル) である。最初の初期化子は配列の 1 番目の初期値として、2 番目の初期化子は第 2 の初期値として、... 、というように格納される。

例 3.2 1 次元配列宣言の初期化を、

```
int a[3] = {4,-2,12};
```

としたとき、{4,-2,12} が初期化子である。

```
int a[3];

a[0] = 4;
a[1] = -2;
a[2] = 12;
```

と同じである。

配列のサイズ (大きさ) が指定されない配列を初期化すると、そのサイズは初期化子の数になる。例えば、

```
int a[] = {4,-2,12};
```

は、初期化子が 3 つあるため、a を 3 つの要素を持つ int 型 1 次元配列として定義し、初期化する。したがって、例 3.2 の配列に一致する。

演習 3.2 配列 a が、

```
double a[] = {1.3, 2.5, -4.3, 3.2};
```

と初期化されているとき、
(1) a[1] の値は何か。
(2) a[4] はどうか。

初期化子に配列のサイズよりも少ない値を与えると、残りには 0 が入る。

例 3.3

```
int a[5] = {4,-2,12};
```

は

```
int a[5] = {4,-2,12,0,0};
```

と同じである。

```
1  /* array-initialization.c */
2  #include <stdio.h>
3
4  int main(void)
5  {
6    int i;
7    int a[5]={4,-2,12};
8
9    for(i = 0;i < 5;i++)
10     printf("&a[%d]=%p a[%d]=%d\n",i,&a[i],i,a[i]);
11
12   /* 次の 2 行は不適切 */
13   i = 5;
14   printf("&a[%d]=%p a[%d]=%d\n",i,&a[i],i,a[i]);
15
16   printf(" &i=%p i=%d\n",&i,i);
17
18   return 0;
19 }
```

1. 7 行目で初期化を行っている。

2. 9-10 行目で、a[0],...,a[4] のアドレスと値を出力している。
3. 変換指定子%p はアドレス (ポインタ型オブジェクト) を出力する。東女の cc では 16 進数で表示される。
4. 13 行目で、a[5] のアドレスと値を出力している。
5. 15 行目で、i のアドレスと値を出力している。a[5] と i が同じアドレスであることが分かる。

出力結果は次のようになる。

```
~/comp3a$ cc -o array-initialization -Wall array-initialization.c
~/comp3a$ ./array-initialization
&a[0]=0xbffff3f8  a[0]=4
&a[1]=0xbffff3fc  a[1]=-2
&a[2]=0xbffff400  a[2]=12
&a[3]=0xbffff404  a[3]=0
&a[4]=0xbffff408  a[4]=0
&a[5]=0xbffff40c  a[5]=5
   &i=0xbffff40c     i=5
```

配列とアドレスの関係は次節ポインタのところで詳しく学ぶ。

3.1.4　1 次元配列の例 (最大値)

例 3.4 N 個のデータ a[0],a[1],a[2],...,a[N-1] の最大値 a[max] は、

1. i=0 のときは max=0 とする。
2. i=1,...,N-1 に対し以下を繰り返す。
   a[max]< a[i] ならば max = i とし、そうでなければ max はそのままとする。
3. このときの a[max] が a[0],a[1],a[2],...,a[N-1] の最大値。

として求める。(最大値を求めるアルゴリズム)

アルゴリズム

アルゴリズム (algorithm) とは、次の 5 条件を満たす特定のタイプの問題を解決するための一連の作業手順を定める有限個の規則である。

1 有限性 (Finiteness) 有限回のステップで終了する。

2 明確性 (Definiteness) 各ステップは明確かつ曖昧さがなく定義されている。

3 入力 (Input) 0 個以上の入力を持つ。入力とは、アルゴリズムが開始する前に最初に与えられるか、実行中に動的に与えられる量である。

4 出力 (Output) 1 個以上の出力を持つ。出力とは、入力との間に指定された関係を持つ量である。

5 実効性 (Effectiveness) それぞれのステップが実行可能である。

double 型データを入力し、Ctrl+d で入力を終了し、入力した数の最大値を求めるプログラムを、配列を用いて作る。データの数は 100 以下であればいくつでもよく、最大値を与えるデータが何番目のデータかも分かる。

```
1  /* max-array.c */
2  #include <stdio.h>
3  #define N 100
4
5  int main(void)
6  {
7    int i,imax;
8    double a[N];
9
10   i = 0;
11   while(scanf("%lf",&a[i]) != EOF){
12     if (i == 0 || a[imax] < a[i]){
13       imax = i;
14     }
15     i++;
16   }
17
18   printf("最大値は a[%d]=%g\n",imax,a[imax]);
19
20   return 0;
21 }
```

3 行目 コンパイルに先立ち N を 100 で置き換える。識別子 N をマクロ名 (macro name) という。

7 行目 int 型オブジェクト imax は、 a[imax] が最大値を与えるような添字である。

8 行目 double 型配列 a を宣言している。サイズは N すなわち、100 である。

11 行目 while(scanf("%lf",&a[i]) != EOF) は scanf の返却値が EOF でないあいだ繰り返す。

12-14 行目 a[i] がそれまでに入力したデータの中で最大値のときは、imax に i の値を代入する。

18 行目 何番目のデータが最大を与え、最大値はいくつであるかを表示する。

scanf() 関数と EOF

1. scanf() はキーボードから書式付きで文字列や数値を入力する関数である。返却値は整数で、正常に入力したときは入力した項目数が返り、1 つも入力できないときは EOF(End of File) を返す。東女の cc では Ctrl+d で入力終了となり、EOF を返す。
2. EOF はヘッダ stdio.h に規定されたマクロで、型は int、負の値をもつ整数定数式 (東女の cc では EOF は-1) に展開する。
3. max-array.c の 11 行目の関数呼び出し式 scanf("%lf",&a[i]) はキーボードから double 型データを入力すると a[i] に格納され、返却値 1 を返す。このとき、論理式 scanf("%lf",&a[i]) != EOF の値は 1【真】であるので、while 文のループ本体 (12-16 行目) が実行される。
4. Ctrl+d を入力すると a[i] にはなにも格納されず、EOF を返す。このとき、論理式 scanf("%lf",&a[i]) != EOF の値は 0【偽】なので、while 文の次の文 (18 行目) が実行される。

3.1.5　1 次元配列の例 (エラトステネスの篩)

例 3.5 1000 以下の素数 (2,3,5,7,11,...,997) を求める。初めにエラトステネスのふるい (Eratosthenes's sieve) について説明する。1 から 1000 までの整数を用意し、素数でない数を消していく。

1. 1 は素数でないので消す。
2. 2 は素数として残し、2 の倍数 (4,6,8,...,1000) を消す。
3. 残っている 2 より大きい最小数 3 は素数として残し、3 の倍数 (9,15,21,...,999) を消す。
4. ...
5. 素数 p の倍数を消したとき、p より大きい最小数が $\sqrt{1000}$ を越えたら、残りすべては素数である。

このアルゴリズムを簡素化し、

1. 1 は素数でないので消す。
2. 2,3,4,...,100 に対し、1000 以下の倍数を消す
3. 残った数が素数である

というアルゴリズムに基づきプログラミングする。配列 num の要素が num[i]==1 のときは i は素数、num[i]==0 のときは i は素数でないとする。

```
/* sieve.c */
#include <stdio.h>
#include <math.h> #define N 1000

int main(void)
{
  int num[N+1];
  int i,j;
  true=1;

  num[1]=0;
  for(i=2;i<=N;i++)
    num[i]=1;
  for(i=2;i<= ceil(sqrt(N));i++)
    for(j=2;j<=N/i;j++)
      num[i*j]=0;
  for(i=2;i<=N;i++)
    if (num[i])
      printf("%4d",i);
  printf("\n");

  return 0;
}
```

3.1.6　引数に配列を持つ関数

int 型配列を引数に持つ関数の仮引数は、int a[] または int *a(int 型へのポインタ) とし、実引数は配列名 a とする。詳細はポインタおよび参照渡しのところで述べる。

例 3.6 max-array.c(例 3.4) の最大値を求める部分を関数にする。

```
1   /* max-array-func.c */
2   #include <stdio.h>
3   #define N 100
4   int max(double a[],int n);
5
6   int main(void)
7   {
8     int n,nmax;
9     double a[N];
10
11    n = 0;
12    while(scanf("%lf",&a[n]) != EOF)
13      n++;
14    nmax = max(a,n);
15
16    printf("最大値は a[%d]=%g\n",nmax,a[nmax]);
17
18    return 0;
19  }
20
21  int max(double a[],int n)
22  {
23    int i,imax;
24
25    imax = 0;
26    for(i = 1;i < n;i++){
27      if (a[imax] < a[i])
28        imax = i;
29    }
30
31    return imax;
32  }
```

4 行目は関数 max() の関数原型である。

21-32 行目で定義した関数 max() は double 型配列と int 型オブジェクトを引数に持つ。

max() の第一引数の仮引数はサイズを指定しない double 型配列、実引数は配列の識別子【配列名】である。第二引数は配列のサイズである。

main() 関数の 14 行目 max(a,n) は関数 max() を呼び出している。返却値 imax が nmax に代入される。

例 3.7 max-array-func.c(例 3.6) の 8 行目で宣言した識別子 n は main() 関数の中だけで有効であるので、関数 max() で n の値が利用できるように、14 行目で実引数として値を渡した。

int 型オブジェクト n をすべての関数 (main() 関数と max() 関数) の外側で宣言するとファイル全体で有効になるので、n の値を関数 max() に渡さなくて良くなる。

```
1  /* max-array-func1.c */
2  #include <stdio.h>
3  #define N 100
4  int max(double a[]);
5  int n;
6
7  int main(void)
8  {
9    int nmax;
10   double a[N];
11
12   n = 0;
13   while(scanf("%lf",&a[n]) != EOF)
14     n++;
15   nmax = max(a);
16
17   printf("最大値は a[%d]=%g\n",nmax,a[nmax]);
18
19   return 0;
20 }
21
22 int max(double a[])
23 {
24   int i,imax;
25
26   imax = 0;
27   for(i = 1;i < n;i++){
28     if (a[imax] < a[i])
29       imax = i;
30   }
31
32   return imax;
33 }
```

5 行目で宣言した int 型オブジェクト n の識別子 n はファイル有効範囲を持つ。
22-33 行目で定義した関数 max() は double 型配列のみを引数に持つ。

有効範囲 (スコープ)

1. オブジェクト、関数などの識別子は宣言の位置により使用可能な範囲が定まる。使用可能な範囲を有効範囲 (scape) といい、識別子は有効範囲にあるとき可視 (visible) という。
2. 宣言がどのブロック (中括弧 { と } で囲まれた部分) またはどの関数定義の仮引数並びの外側に現れる場合、その識別子は**ファイル有効範囲** (file scope) を持ちソースファイル全体で可視である。【グローバル変数】
3. 宣言がブロックまたは関数定義の仮引数ならびに現れる場合、その識別子は**ブロック有効範囲** (block scope) を持ちその範囲は対応するブロックである。【ローカル変数】

max-array-func.c の 8 行目の識別子 n はブロック有効範囲を持ち、7 行目から 19 行目までで可視である。一方、max-array-func1.c5 行目の識別子 n はファイル有効範囲を持ち、ソースファイル全体 (1 行目から 33 行目) で可視である。
max-array-func1.c9 行目の識別子 nmax は、ブロック有効範囲を持ち、main() 関数本体 (8 行目から 20 行目まで) で可視である。max-array-func1.c24 行目の識別子 imax は、ブロック有効範囲を持ち、関数 max() 本体 (23 行目から 33 行目まで) で可視である。

演習 3.3 n 個のデータ $x_1, x_2, \ldots, x_n$ の平均 m、標準偏差 σ(シグマ) は次式で定義される。

$$m = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n} - m^2} = \sqrt{2\text{ 乗の平均} - \text{平均の 2 乗}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2}$$

例えば、2.5、7.3、1.4 の平均は 3.73、標準偏差は 2.56 である。

1. double 型配列にデータを入力し平均と標準偏差を求めるプログラムを while 文を用いて作れ。入力の終了は Ctrl-d で行うものとする。
2. 平均を求める関数 average() と標準偏差を求める関数 stdevp() を作り 1 と同様のことを行え。

3.1.7　1 次元配列の例 (整列法)

データを順に並べ替える方法を整列法 (sorting method) という。小さい順に並べることを昇順、大きい順に並べることを降順という。ここでは、昇順を扱うが、降順の場合は不等号の向きを逆にすれば良い。

整列法には多くの方法が知られているが、最も簡単な方法の一つである選択整列法 (selection sort) を取り上げる。

配列の最小の要素を見つけ、最初の位置にある要素と交換する。次に、2 番目に小さい要素を見つけ、2 番目の位置にある要素と交換する。以下同様にして、配列全体が整列されるまで繰り返す。

5, 4, 2, 6, 1, 3 に選択整列法を適用する過程を示す。◯ のついた数が選択された数である。

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

`a[0],...,a[n-1]` に対する選択整列法のアルゴリズムを以下に示す。
`i=0,...,n-1` に対し、

1. `a[i],...,a[n-1]` の最小値を `a[min]` とする。
2. `a[i]` の値と `a[min]` の値を交換する。(`a[i]` が最小値のときは、何も行わない。)

このとき、`a[0],...,a[n-1]` は昇順に並んでいる。

プログラムは以下のようになる。

```
1  /* 選択整列法 */
2  /* selection-sort.c */
3  #include <stdio.h>
4  #define N 100
5  void selection(int a[],int n);
6
7  int main(void)
8  {
9    int n,i,a[N];
10   n = 0;
11   while(scanf("%d",&a[n]) != EOF)
12     n++;
13
14   selection(a,n);
15
16   /* 結果出力 */
17   for(i = 0;i < n;i++)
18     printf("%3d ",a[i]);
19   printf("\n");
20
21   return 0;
22 }
23
24 void selection(int a[],int n)
25 {
26   int i,j,min,t,count = 0;
27   for(i=0;i < n;i++){
28     min = i;
29     for(j = i + 1;j < n;j++){
30       count++;
31       if (a[j] < a[min]) min = j;
32     }
33     t = a[min];
34     a[min] = a[i];
35     a[i] = t;
36     /* 途中結果出力開始 */
37     for(j = 0;j < i;j++)
38       printf("   ");
39     for(j = i;j < n;j++)
40       printf("%3d",a[j]);
41     printf("\n");
42     /* 途中結果出力終了 */
43   }
44   printf("比較回数 %d\n",count);
45 }
```

27 行目から 43 行目の for 文では、i=0,...,n-1 について i+1 番目に小さい要素を探し、a[i] と交換している：

1. 28 行目から 32 行目では a[i] から a[n-1] のなかの最小値 a[min] を求めている。
2. 33 行目から 35 行目では a[i] と a[min] を交換している。
3. 36 行目から 42 行目では a[i] と a[min] を交換した直後の配列の値を表示している。37 行目と 38 行目では、すでに確定している値を空白で表している。

整列法のアルゴリズムの優劣は比較回数または最大交換回数で評価することが多い。関数 selection() には比較回数を数え出力するルーチンを含めてある。n 個のデータの選択整列法による比較回数は $(n-1)+(n-2)+\ldots+1 = n(n-1)/2$ である。最大交換回数は $n-1$ 回である。たとえば、2, 3,..., n-1, 1 を小さい順に整列する場合である。

単に並べ替えるだけなら、オブジェクト `count` に関する部分 (26 行目 “count = 0”、30 行目)、途中結果出力の開始から終了まで (36-42 行目)、と比較回数の出力 (44 行目) は不要である。

例 3.8 selection-sort.c では `a[i]` の値 (キーという) だけを並べ替えたが、

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
a[4]	a[1]	a[2]	a[3]	a[0]	a[5]
	a[2]	a[1]	a[3]	a[0]	a[5]
		a[5]	a[3]	a[0]	a[1]
			a[1]	a[0]	a[3]
				a[0]	a[3]

と添字も必要になることがある。このときは添字の配列 index[] を作りキーが交換される際、index[] の値も交換する。上の例の場合、整列したあとは `4 == index[0]`、`2 == index[1]`、`5 == index[2]`、`1 == index[3]`、`0 == index[4]`、`3 == index[5]` となる。

selection-sort.c の main() 関数で、

```
for(i = 0;i < N;i++)
  index[i] = i;
```

と初期値を与えておき、関数 selection() の仮引数に `int index[]` も加え、

```
t = a[min];
a[min] = a[i];
a[i] = t;
```

に続き、

```
t = index[min];
index[min] = index[i];
index[i] = t;
```

を加える。

演習 3.4 例 3.8 のプログラム selection-sort-index.c を完成させよ。途中結果の出力は必要ない。

演習 3.5 double 型データをキーボードから入力し、選択整列法により小さい順に並べ替えるプログラム selection-sort-double.c を作れ。

演習 3.6 複数のアスキー文字 (英字、数字、記号) を 1 文字づつキーボードから入力し、選択整列法でアスキーコード順に並べ替えるプログラム selection-sort-ascii.c を作れ。
Hint.

- int a[] を char a[] に、%d を%c に置き換える。
- `scanf("%c",&a[n])` で文字を入力すると `a[n+1]` に lf(0x0a) が入力されるので、selection-sort.c の 11-12 行目を

```
while(scanf("%c",&a[n]) != EOF){
  n++;
  getchar();
}
```

とする。このとき getchar() が lf(0x0a) を読み取る。getchar() 関数はキーボードから文字を unsigned char 型として読み込み int 型に変換する。読み取りに失敗したときは EOF を返す。

演習 3.7 挿入整列法 (insertion sort) は、最初の 2 つを整列する。次に最初の 3 つを整列する。3 つ目が最初の二つの中に挿入される。次に最初の 4 つを整列する。以下同様にして、配列全体が整列されるまで繰り返す。

5, 4, 2, 6, 1, 3 に挿入整列法を適用する過程を示す。◯ のついた数を挿入する。

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

挿入整列法 insertion-sort.c のプログラムを作れ。

演習 3.8 (オプション) n 個のデータの挿入整列法による比較回数を求めよ。

3.2 文字列

char 型配列は文字列に用いられる。日本語文字列、char 型へのポインタや文字列操作関数を用いた文字列の扱いは次節以降で学ぶ。

3.2.1 文字列リテラル

二重引用符”で囲まれた 0 個以上の文字の列を 単純文字列リテラル (character string literal)【文字列リテラル (string literal)】という。ただし、二重引用符とバックスラッシュはそれぞれ拡張表記 \" と \\ で表わす。

例えば、"abc"、"Hello world\n"は文字列リテラルである。

文字列リテラルはコンパイルの際、文字のならびの最後にナル文字\0 を付加する。プログラム開始処理の際 1 回だけ初期化され、実行の最後まで値は変わらない。(静的記憶域期間を持つという。)

例えば、文字列リテラル"abc"は、メモリに

a	b	c	\0

と格納される。16 進数では

0x61	0x62	0x63	0x0

なのでビット列としては

01100001	01100010	01100011	00000000

である。

3.2.2 文字列

文字列 (string) とは、連続した文字の列で最初のナル文字\0 で終わるものをいう。最初のナル文字は文字列に含まれる。

演習 3.9 次は文字列か。

1. | a | b | c | d |
2. | a | b | c | \0 |
3. | a | b | c | \0 | d | \0 |
4. | a | b | c | 0 |

文字列の長さは、ヌル文字\0 に先行する文字の個数とする。例えば、文字列 | a | b | c | \0 | の長さは 3 である。

例 3.9 1. "abc" は長さ 3 の文字列 | a | b | c | \0 | を表す文字列リテラルである。

2. "Hello world\n" は長さ 12 の文字列 | H | e | l | l | o | | w | o | r | l | d | \n | \0 | を表す文字列リテラルである。o の次の空白は 16 進数の 0x20(ビット列は 0010 0000) である。

文字列は char 型配列を文字列リテラル (配列のサイズは文字列の長さ +1) で初期化して表すことができる。文字列を printf() 関数で表示する際の変換指定子は%s である。

```
1  /* string.c */
2  #include <stdio.h>
3
4  int main(void)
5  {
6    char str[]="abc"; /* 文字列リテラルによる初期化 */
7    int i=0;
8
9    while(str[i])
10     i++;
11
12   printf("str = %s\n",str);
13   printf("文字列の長さ = %d\n",i);
14
15   return 0;
16 }
```

9 行目から 10 行目の while 文は、str[i] がヌル文字のときは値が 0 になるので、str[i] がヌル文字'\0' になるまで i++が実行される。str[i] == '\0' のとき、i が文字列の長さになる。

3.2.3 文字列の初期化

文字列を char 型配列で初期化するには文字列リテラルによる方法以外に、配列の各要素に char 型データを与えるいくつかの方法がある。(char 型へのポインタとして文字列を扱うこともできるが、これについては次節以降で扱う。)

```
1  /* string-initialization.c */
2  #include <stdio.h>
3
4  int main(void)
5  {
6    char str1[]={'a','b','c','\0'}; /* 文字定数 */
7    char str2[]={'\x61','\x62', '\x63','\x0'}; /* 16 進逆斜線表記による文字定数 */
8    char str3[]={0x61,0x62,0x63,0x0}; /* 16 進定数 */
9    char str4[]={97,98,99,0}; /* 10 進定数 */
10   char str5[]="abc"; /* 文字列リテラル */
11
12   printf("str1 = %s\n",str1);
13   printf("str2 = %s\n",str2);
14   printf("str3 = %s\n",str3);
15   printf("str4 = %s\n",str4);
16   printf("str5 = %s\n",str5);
17
18   return 0;
19 }
20
```

3.2.4 文字列の出力

3.2.4.1 文字列として出力 文字列 str を出力するには `printf()` 関数で変換指定子`%s` を用いる。

文字列の出力にはライブラリ関数 `puts()` を使うこともできる。`puts(str)` は、str が指す文字列を出力する。出力の最後に改行文字を追加する。文字列の終端のナル文字は出力しない。

演習 3.10 string.c の `printf()` を `puts()` を用いて書き直せ。

3.2.4.2 文字単位で出力 文字列を文字単位で出力することもできる。

```
for(i = 0;str[i];i++)
  printf("%c",str[i]);
```

str[i] がナル文字のときは値が 0 になるのでナル文字の直前までが 1 文字ずつ出力される。

```
i = 0;
while(str[i]){
  printf("%c",str[i]);
  i++;
}
```

とすることもできる。('\0' は値が 0(偽) なので)

演習 3.11 文字列リテラル`"abc"`を文字単位で出力するプログラムをつくれ。

3.2.5 文字列の入力

文字列 str をキーボードから入力するには scanf("%s",str) とする。%s は文字列を入力する変換指定子である。scanf("%s",str) は、空白が来ると読み込みを止めるので"This is a string."を入力しても、"This"しか読み込まれない。

ライブラリ関数 gets() を使って、gets(str); とすると空白も読み取れる。gets(str) は入力された文字列を str が指す配列に格納する。改行文字を受け取ったときに文字の読み取りは終了する。読み取った改行文字を捨て、配列に格納した文字の直後にナル文字を書く。gets() を用いるとコンパイラは警告[5]を発する。

```
1  /* string-input.c */
2  #include <stdio.h>
3
4  int main(void)
5  {
6    char str[100];
7
8    printf("文字の列 (99 文字以内) を入力してください。\n");
9    gets(str);
10
11   printf("ただいま入力した文字の列は \n");
12   printf("%s\n",str);
13   printf("です。\n");
14
15   return 0;
16 }
```

演習 3.12 string-input.c の 9 行目を scanf() 関数を用いて書き直せ。空白を含む文字列を入力するとどうなるか。

演習 3.13 キーボードから文字列を入力し、入力した文字列の長さを表示するプログラムを for 文または while 文を用いて作れ。たとえば、"This is a string." と入力すると"17 文字" と出力する。

3.3 多次元配列

3.3.1 2 次元配列

2 次元配列の宣言は

```
要素型 識別子 [式 1][式 2];
```

である。

例 3.10 配列宣言が

```
int a[2][3];
```

となっているとき、要素型は int、識別子 a、式 1 は整数定数 2、式 2 は整数定数 3 である。配列 a は 2 つの要素の配列であり、各要素は int 型の要素数 3 の 1 次元配列である。(2×3 の 2 次元配列ということがある。) 図式化すると

[5] gets() にはこれから読み込むバッファの大きさを伝えることができないので、大きな文字列を読み込むとバッファがあふれセキュリティホールとなる。string-input.c の 9 行目を fgets(str, sizeof str, stdin); に書き直すと問題は生じない。

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

となるが、メモリ上は、長さ 6 の 1 次元配列

アドレス低い ⟷ アドレス高い

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

として格納されている。配列は、行優先の順序 (最後の添字が最も速く変る) で格納する。

長方形に配置したデータの横のならびを行といい、縦のならびを列という。

例 3.11 200 人の生徒の、英語、数学、国語の成績は int 型の 2 次元配列 mark[200][3] を用いて、

	英語	数学	国語
学籍番号 1 番	mark[0][0]	mark[0][1]	mark[0][2]
学籍番号 2 番	mark[1][0]	mark[1][1]	mark[1][2]
⋮			
学籍番号 200 番	mark[199][0]	mark[199][1]	mark[199][2]

とすることができる。学籍番号 100 番の生徒の 3 科目の合計点は、

```
mark[99][0]+mark[99][1]+mark[99][2]
```

で与えられる。

200 人の生徒の英語, 数学, 国語の試験の点数を入力し、全員の教科ごとと合計点を出力するプログラム。

```
/* mark.c */
/* 成績の処理 */
#include <stdio.h>
#define N 200 /* 人数 */
#define M 3 /* 科目数 */

int main(void)
{
  int mark[N][M],sum[N] = {0};
  int i,j;

  printf("各生徒に対し英語，数学，国語の順に入力してください。\n");

  for(i = 0;i < N;i++){
    printf("%d 番の英語，数学，国語の点:  ",i);
    scanf("%d %d %d",&mark[i][0],&mark[i][1],&mark[i][2]);
    for(j = 0;j < 3;j++)
      sum[i] += mark[i][j];
  }
  printf("\n");

  printf("%4s %5s %5s %5s %5s\n\n","番号","英語","数学","国語","合計");
  for(i = 0;i < N;i++)
    printf("%4d %5d %5d %5d %5d\n",i,mark[i][0],mark[i][1],mark[i][2],sum[i]);

  return 0;
}
```

例 3.12 数値計算においては、数列や数ベクトルは double 型の 1 次元配列、行列は double 型の 2 次元配列を用いる。

3.3.2 2 次元配列の初期化

2 次元配列の初期化は、

型 識別子 [式 1][式 2]={ 初期化子並び };

とする。

例 3.13 2 次元配列の初期化を

```
int a[2][3] = {
  {1,2,3},
  {4,5,6}
};
```

としたとき、{1,2,3} は a の第 0 行を初期化し、{4,5,6} は、a の第 1 行を初期化する。したがって、a[0][0] は 1、a[0][1] は 2、...、a[1][2] は 6 というようになる。

```
int a[2][3] = {
  1,2,3,4,5,6
};
```

と {1,2,3}, の波括弧とその後の改行を省略しても同じ結果を与える。

配列の式 1(行の数) は省略できるが、配列の式 2(列の数) は省略できない。int a[2][3]; はサイズが 3 の 1 次元配列 2 個を要素とする 2 次元配列であるので、行の数 (1 次元配列の数)2 は省略して

```
int a[][3] = {
  1,2,3,
  4,5,6
};
```

としてもよい。(大きさが 3 の 1 次元配列が 2 個あることはコンパイラが分かる。)

初期化したとき、初期化子の数が配列の大きさより少ないときは、残りの要素の初期値は 0 となる。

```
int a[4][3] = {
  1,2,3,
  4,5,6
};
```

は、

```
int a[4][3] = {
  {1,2,3},
  {4,5,6},
  {0,0,0},
  {0,0,0}
};
```

と同じである。

配列の大きさより初期化子の数が多いときはエラーとなる。

3.3.3 文字列の配列

文字列は char 型の配列で表せるので、文字列の配列は 2 次元 char 型配列で表現できる。

演習 3.14 string-initialization.c を 2 次元 char 型配列を用いて書き換えよ。
Hint: char str[5][4] を用いる。5 は文字列の個数、4 は文字列の長さ +1 である。

3.3.4 2 次元配列を引数に取る関数

配列 `int a[4][3]` を関数の引数にするには、仮引数として `int a[][3]` と書く。

演習 3.15 `mark.c` の `main()` 関数では入力だけを行い、合計点の計算と出力は関数 `output()` で行うように書き換えよ。