配列 (基礎)

山本昌志*

2006 年 12 月 15 日

概 要

同じ型の大量のデータを格納する場合に便利な配列と呼ばれるデータ構造を学習する．ここでは，配列の宣言の仕方とデータの入出力方法を主に説明する．

1 本日の学習内容

本日は，配列というデータ構造について，学習する．いままで，諸君が知っているデータ構造は，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—メモリーの領域—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—メモリーの領域—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—a や w—を指定することで，データにアクセスできる．

ここでは，もっと進んだデータ構造を学習する．それは，配列と呼ばれるもので，

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

のように宣言し，名前と自然数によりデータにアクセスできる．この配列の使い方を理解することで，とんでもなく大量のデータを取り扱うことができるようになる．

2 配列の必要性と基本的な使い方

2.1 たくさんの値を記憶する必要性

コンピュータでは，データを記憶する場所には，プログラマーが名前を付ける必要がある．そうしないと，プログラマーは記憶したデータを取り扱うことができない．これまでに，諸君は単純型と言われるデー

*独立行政法人　秋田工業高等専門学校　電気情報工学科

タ構造を学習した．それを使うためには，記憶するデータの型と変数名をプログラムの最初に記述する．次のようにである．

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

このようにいちいち名前を指定する方法だと，大量のデータを処理することは不可能である．たとえば，100万個の名前を付けるだけで，大変である．困った，さあどうするか?

2.2 配列の宣言

先ほどのデータ構造—単なる変数—で，大量のデータを扱うことは不可能である．なぜならば，単なる変数で一度に確保できるメモリーの領域は 1 個であるため，全てのデータに異なった名前を付けることは不可能となるからである．ようするに，100 万個のデータを扱う場合，それだけの変数名を用意するのはナンセンス．そこで，複数の同じ型のデータを取り扱うために，配列というデータ構造が発明された．

これは，同じ型のデータを任意の個数宣言し，配列名と自然数でアクセスすることができるようにしたものである．配列宣言を使う場合の宣言は，型名と配列名，そしてサイズを書く．

書式 (配列の宣言方法)

```
データ型名 配列名 [サイズ];
データ型名 配列名 [サイズ][サイズ];
データ型名 配列名 [サイズ][サイズ][サイズ];
```

具体的には，次のようにする．

```
int i[10], j[100][100];
```

こうすると，

- 配列名 i の整数型のデータ領域が 10 個用意される．用意されるデータ領域は，i[0]〜i[9] である．
- 配列名 j の整数型のデータ領域が 10000 個用意される．用意されるデータ領域は，j[0][0]〜j[99][99] である..

となる．C 言語では配列の添字 (インデックス) は，ゼロからはじまることに注意が必要である．

C 言語では，配列のデータは連続したメモリー領域に格納される．すなわち，先ほどの例の場合，図 1 のようにメモリー領域が確保される．連続したメモリー領域をつかうので，配列へのデータの出し入れは高速にできる．

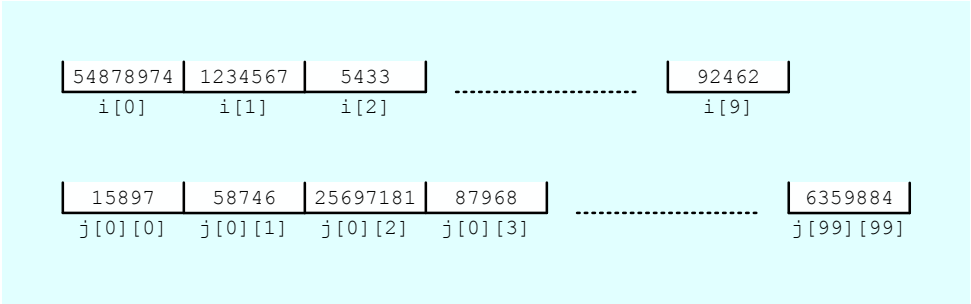

図 1: 配列のイメージ．データを入れる箱がいっぱいある．ただしひとつの箱の大きさは全て同じで整数を格納する．

2.3 配列の初期化

配列のサイズが小さい場合，宣言と同時に初期ができる．

```
int hoge[3]={111,222,333};
int fuga[2][2]={{111,222},{333,4444}};
```

このようにすると，

```
hoge[0]=111      hoge[1]=222      hoge[2]=333
fuga[0][0]=111   fuga[0][1]=222   fuga[1][0]=333   fuga[1][1]=4444
```

と初期値が設定される．配列よりも初期値が少ない場合には，残りのゼロに初期化される．多い場合にはエラーとなる．

2.4 配列へのアクセス

配列では，配列名と添え字 (インデックス) を指定する—たとえば `i[3]` や `j[25][49]`—ことにより，記憶領域から値 (データ) を入出力できる．

```
i[3]=5;          /* 配列 i[3] に 5 を代入  */
c=j[25][49];     /* 配列 j[25][49] の値を変数 c へ代入 */
```

配列全体をコピーする場合，要素毎に代入文を実行する必要がある．例えば，サイズ 100×200 の配列 `hoge[][]` と `fuga[][]` があるとする．`hoge[][]` のデータを `fuga[][]` にコピーする場合，次のようにする．

```
for(i=0; i<=99; i++){
  for(j=0; i<=199; i++){
    fuga[i][j]=hoge[i][j];
  }
}
```

2.5 注意

配列を使う場合，以下のような間違いが多い．気を付けよ．

- 配列の宣言の大括弧内 [] の値はサイズである．そして，配列の添字はゼロからはじまる．多くの間違いは，

  ```
  int hoge[100];
  ```

  のように配列を宣言し，

  ```
  hoge[100]=2468;
  bar=hoge[100];
  ```

  のように，i[100] を使おうとする．範囲は，i[0]–i[99] までである．

- C 言語のプログラムの動作時，配列の添字 (インデックス) の範囲がチェックされない．宣言した範囲を越えて，読み書きができる可能がある．先ほどの例では，hoge[99] が使用可能範囲であるが，hoge[345] も読み書きができることがある．他のプログラムが使っているメモリー領域にアクセスすることになる．するとプログラムがクラッシュする可能性がある．添字の範囲のチェックはプログラマーの責任で，OS は関知しない．悪意のあるプログラマー—クラッカー— はこの性質を利用して，悪さをするプログラムを作ることがある．

- 配列の要素にアクセスするときの添字は整数である．倍精度実数は使えない．

3 配列の例

3.1 1 次元配列

添字 (インデックス) がひとつのものを一次元配列と言う．例えば，

```
int ab[1000000], cd[1000];
double xy[1000000], z[60];
```

である．これで，

- 整数が格納できる ab[0]〜ab[999999] の 100 万個の記憶場所
- 整数が格納できる cd[0]〜cd[999999] の 100 万個の記憶場所
- 倍精度実数が格納できる xy[0]〜xy[999999] の 100 万個の記憶場所
- 倍精度実数が格納できる z[0]〜xy[59] の 60 個の記憶場所

が確保できた．

このデータ構造で，個々の内容にアクセスするためには，配列名と自然数である添え字 (インデックス) を指定する．次のようにする．

```
a = ab[12345];
ab[23456] = 654321;
scanf("%d",&ab[34567]);
printf("%d\n",ab[45678]);
```

今まで，使ってきた単純型と同じである．

大量のデータを配列を使って取り扱うとき，繰り返し文 (ループ) を使うのが普通である．たとえば，つぎのようにすると，短い文で大量のデータを処理するプログラムが書ける．

```
for(i=0; i<=360; i++){
  s[i] = sin(i*M_PI/180);
  c[i] = cos(i*M_PI/180);
}
```

これで，三角関数の値が 0〜360 度まで計算できる．

3.2 2 次元配列

1 次元配列は，配列名とひとつの添字 (自然数) でデータにアクセスする．それに対して，配列名と 2 つの添字を用いるものを 2 次元配列と言う．

たとえば，1 日の最高気温のデータを年間にわたって格納したい場合，

```
double max_temp[13][32];
```

というように配列を宣言する．これで，max_temp[0][0]〜max_temp[12][31] まで使える．最初の添字 (インデックス) が月を表し，次の添字が日を表す．分かり易いデータ構造となっている．これに，日々の最高気温をキーボードから入力する場合，次のようにプログラムを書く．

```
double max_temp[13][32];
int dates[13] = {0,31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int i,j;

for(i=1; i<=12; i++){
  for(j=1; j<=dates[i]; j++){
    printf("%d 月%d 日の最高気温\t", i, j);
    scanf("%lf", &max_temp[i][j]);
  }
}
```

配列 dates[] には，月の日数を入れておく．閏年は考えていない．2 次元配列にアクセス屡ときには，二重ループが使われることが多い．

3.3 多次元配列

配列の添字の数を次元と言う．それが 1 個だと 1 次元配列，2 個だと 2 次元配列，それ以上のものも考えられる．先程の最高気温を年月日で処理する場合，3 次元配列が適当であろう．

4 プログラム作成の練習

[**練習 1**] 1001 個の配列を用意して，そこに 0〜10000 の整数を格納せよ．そして，配列に格納している値を合計せよ．

[**練習 2**] 10000×10000 の乗算の値を格納した配列を作成せよ．その表を利用して，$m \times m$ のように，同じ整数同士の演算結果を表示せよ．

[**練習 3**] 10 個の整数をキーボードから入力して，入力した順序と逆に表示するプログラムを作成せよ．

[**練習 4**] 付録付録 B に示したエラトステネスのふるいを使って，100000 までの素数を求め表示せよ．

5 課題

試験明け後の最初の講義の日 (12 月 22 日) の AM8:45 までに，以下の課題をレポートとして提出すること．表紙等は，いつもの通り．表紙のタイトルは「配列 (基礎)」とすること．

[**問 1**] (復予) 教科書 p.204–245 を 3 回，繰り返し読め．

[**問 2**] (復) 以下の動作をおこなうプログラムを作成せよ．

- 10 個の整数をキーボードから入力する．
- 全ての整数が異なれば「全ての整数は異なります」と，同じ整数があれば「同じ整数があります」と表示する．

[**問 3**] (復) 以下の動作をおこなうプログラムを作成せよ．

- 10 個の整数をキーボードから入力する．
- つぎに基準となる整数をキーボードから入力する．
- 基準より大きい整数の数を表示する．

付録 A　データ構造

コンピューターで処理するデータは，全て，数字[1]に置き換えられる．絵や音，あるいは文字なども全て，数字に直して取り扱っているのである．そして，その取り扱う数字の個数は莫大である．たとえば，画素数が 600×800 のカラー画像を考える．

- 一つの画素は，赤 (R) と緑 (G)，青 (B) の強弱を変えることで表現されている．それぞれの色 (RGB) は，0〜255 までの値を取る．
- カラーがを構成するために必要な数の総数は，$3 \times 800 \times 600 = 1440000$ である．

たった，一枚のカラー画に 144 万の数字が必要である．動画になると，これを 1 秒間に 30 回も計算している．驚くべきことである．

画像以外にも，コンピューターが処理するデータは沢山ある．それらは，全て数字となっており，それをどのように表現するかがプログラムのポイントなる．データの表現の方法をデータ構造と言う．このデータ構造をまとめたもの表 1 に示す．これまでは，基本データ型の単純型のみを使ってプログラムを組んできたが，今回で配列を学習したわけである．

諸君は，配列をつかうことで，大量のデータを効率よく扱えるようになったのである．ただ，表を見てわかるようにこれまで学習したものは，まだデータ構造の一部であることが分かる．2 年生以降，これらのデータ構造の詳細を学習することになる．プログラミングの修得の道のりは，長いのである．

表 1: データ構造の種類

データ構造	基本データ構造	基本データ型	単純型	整数型
				実数型
				文字型
				論理型
				数え上げ型
			ポインタ型	
		構造型	配列型	
			レコード型	
		抽象データ型		
	問題向きデータ構造	線形リスト	単純リスト	
			双リスト	
			環状リスト	
		木	二分木	完全二分木
				二分探索木
				バランス木
			多分木	
			バランス木	AVL 木
				B 木
		スタック		
		キュー		

[1]正確に言うと，0 と 1 のビット列

付録Ｂ　エラト ステネスのふるい

「エラト ステネスのふるい」とは素数を求める古典的な方法である．例えば，25 までの素数は次のようにして求める．ただし，1 は素数ではない．

1. はじめに，2〜25 までの整数を用意する．
   2　3　4　5　6　7　8　9　10　11　12　13　14　15　16　17　18　19　20　21　22　23　24　25
2. 最小の素数である左端の 2 を残して，その倍数を消去する．
   2　3　5　7　9　11　13　15　17　19　21　23　25
   これで 2 の処理は終了．
3. つぎに小さい素数である 2 の次の 3 を残して，その倍数を消去する．
   2　3　5　7　11　13　17　19　23　25
   これで 3 の処理は終了．
4. つぎに小さい素数である 3 の次の 5 を残して，その倍数を消去する．
   2　3　5　7　11　13　17　19　23
   これで 5 の処理は終了．
5. 　これを繰り返すと，25 までの素数がのこる．
   2　3　5　7　11　13　17　19　23