2020回
計算機プログラミング演習

第8回

二次元配列

第8回の目標

- 配列の確認とよく使われるテクニック
  - 初期値代入
  - 平均値、最大値と最小値
- 多次元の配列

配列の初期値代入

```
/*
 配列 age[0]~[11] に入ったデータの平均値、最大値と最小値を求めよ。
    なお、配列 age の中のデータは年齢であるので、正の整数である。
*/

#include  <stdio.h>

int  main( void ){

      int   age[12] = { 5, 36, 20, 22, 22, 24, 20,
                        27, 21, 21, 24, 22 };

                  :
          (ここにプログラムを作成する)
                  :
      return 0;

}
```

平均値を求めるには？

①全データの合計を求め、
②データ数で割れば平均が求められる。

dim_test1_sum.c

```
int  sum, ave;
int i;

sum = 0;
for (i = 0; i <= 11; i++) {
     sum = sum + age[i];
}
ave = sum / 12 ;

printf(”平均 %d¥n”, ave);
```

最大値（最小値）を求めるには？

最大値：全データの中でいちばん大きい数

①最大値候補を最初に設定する。
②データ一つずつと最大値候補を比較する。
③最大値候補より大きい数であれば、その数を新しい最大値候補にする。
④全データに対して比較を行い、残った候補が最大値になる。

最小値を求めるには、上記の「最大値」を「最小値」に読み替え、③の部分を「候補より小さい数」に変更すれば良い。

配列の初期値代入

```
/*
 配列 age[0]~[11] に入ったデータの平均値、最大値と最小値を求めよ。
    なお、配列 age の中のデータは年齢であるので、正の整数である。
*/

#include  <stdio.h>

int  main( void ){

      int   age[12] = { 5, 36, 20, 22, 22, 24, 20,
                        27, 21, 21, 24, 22 };

                  :
          (ここにプログラムを作成する)
                  :
      return 0;

}
```

最大値、最小値を求める (1)

dim_test1_max.c

```
int  max ;
int i;

max = 0;
for (i = 0; i <= 11; i++) {
     if ( max < age[ i ] ){
          max = age[ i ];
     }
}

printf(”最大値 %d¥n”, max );
```

最大値、最小値を求める (2)

dim_test1_min.c

```
int  min ;
int i;

min = 0;
for (i = 0; i <= 11; i++) {
     if ( min > age[ i ] ){
          min = age[ i ];
     }
}

printf(”最小値 %d¥n”, mix );
```

正しい結果になりましたか？

最大値、最小値を求める (3)

最大値候補を最初に設定する

候補値の選択に問題が起こる。

(A) 最大値の場合
扱っているデータが年齢であるため、負の数になることはない。
候補値としてゼロを設定 (`max = 0 ;`) しても 最大値は必ずそれ以上に なる。

(B) 最小値の場合
候補値としてゼロを設定 (`min = 0 ;`) するとゼロが最小値として残ってしまう。

適切な最大値候補を最初に設定することが重要！

配列の初期値代入

```
/*
 配列 age[0]~[11] に入ったデータの平均値、最大値と最小値を求めよ。
    なお、配列 age の中のデータは年齢であるので、正の整数である。
*/

#include  <stdio.h>

int  main( void ){

      int   age[12] = { 5, 36, 20, 22, 22, 24, 20,
                        27, 21, 21, 24, 22 };

                  :
          (ここにプログラムを作成する)
                  :
      return 0;

}
```

最大値、最小値を求める (4)

dim_test1_max_min.c

```
int   min, max ;
int i;

min = age[ 0 ] ;
max = age[ 0 ] ;

// 順番は 0 ~ 11 であるが、 0 番を最初の候補にしている
// ので比較範囲は 1 ~ 11 である。

for (i = 1; i <= 11; i++) {
      if ( max < age[ i ] ){
            max = age[ i ];
      }
      if ( min > age[ i ] ) {
            min = age[ i ] ;
      }
}
printf("最小値 %d 最大値 %d¥n", min, max );
```

配列とfor文

```c
/*    hairetsu1.c   */

#include <stdio.h>

int main(void) {

    int sum = 0;
    int array[] = { 102, 673, 899, 547, 987, 123 };

    for (int i = 0; i < sizeof(array) / sizeof(array[0]); i++) {
        sum += array[i];
    }
    printf("合計は%d¥n", sum);
    return 0;
}
```

```
合計は3331
```

実行結果

これまで，配列の宣言と同時に要素数も指定したが，
宣言時に指定しない場合は代入する値の数だけ配列
の要素が生成される

配列とfor文

```
1   /*    hairetsu1.c   */
2
3   #include <stdio.h>
4
5   int main(void) {
6
7       int sum = 0;
8       int array[] = { 102, 673, 899, 547, 987, 123 };
9
10      for (int i = 0; i < sizeof(array) / sizeof(array[0]); i++) {
11          sum += array[i];
12      }
13      printf("合計は%d¥n", sum);
14      return 0;
15  }
16
```

for文の条件式の赤い点線の部分は繰り返しの回数を決めている部分であるが，この部分に“sizeof 演算子”が用いられている．この演算子を用いると変数や配列のメモリサイズが分かる

sizeof（変数名や配列名）

赤い線部分のような記述をしているのは，メモリサイズは配列の要素数に比例するので，全体の配列のメモリサイズを要素1つ分のメモリサイズで割れば，要素数が分かるからである

配列の値を配列に代入

ある配列の値を別の配列の値に置き換える(コピー)こともできる

hairetsu2.c

```c
#include <stdio.h>

int main(void) {

    int array1[] = { 0, 0, 0, 0 };
    int array2[] = { 345, 67, 89, 65, 76 };


    for (int i = 0; i <= sizeof(array1) / sizeof(array1[0]); i++) {

        array1[i] = array2[i];

        printf("array1[%d]= %d¥n", i, array1[i]);

    }

    return 0;
}
```

注意)このままでは強制終了となる

配列の値を配列に代入

ある配列の値を別の配列の値に置き換える(コピー)こともできる

hairetsu2.c

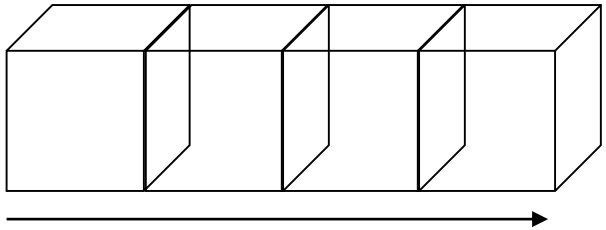
```c
#include <stdio.h>

int main(void) {

    int array1[] = { 0, 0, 0, 0 };
    int array2[] = { 345, 67, 89, 65, 76 };


    for (int i = 0; i < sizeof(array1) / sizeof(array1[0]); i++) {

        array1[i] = array2[i];

        printf("array1[%d]= %d¥n", i, array1[i]);

    }

    return 0;
}
```

```
array1[0]= 345
array1[1]= 67
array1[2]= 89
array1[3]= 65
```

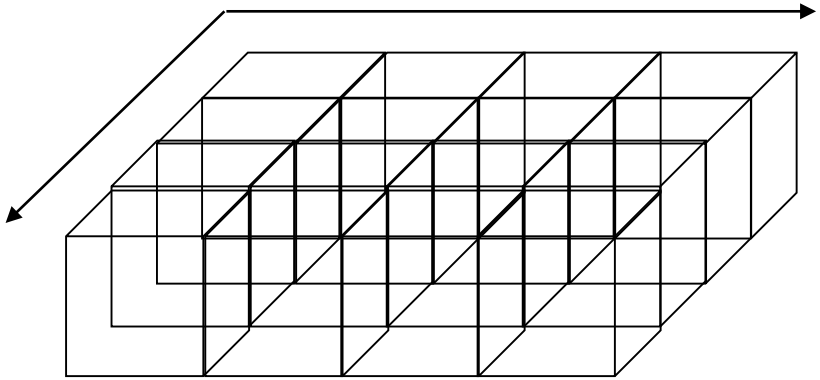
実行結果

ある配列の全ての値を別の配列に代入する場合は,
代入先の要素数に注意すること

２次元配列とは



１次元配列



２次元配列

配列の宣言

型名　配列名 [要素数] [要素数]；
例）int seisu[10] [20]；

配列は多次元化することは可能．ただし，下記の点に注意

- 3次元以上は人にとって理解しにくい
- メモリを消費するので本当に必要か注意すること

二次元配列(1)

- 出席番号１～８の生徒の国語と数学と英語成績

添え字 →	0	1	2	3	4	5	6	7
↓	番号	1	2	3	4	5	6	7	8
0	国語	45	64	58	66	51	76	45	39
1	数学	50	72	92	62	33	88	53	62
2	英語	65	84	43	54	45	65	35	38

データが２次元の行列（３行８列）となっている

二次元配列(2)

生徒の合計と教科の平均を求めるプログラム

```
// two_dim_test.c

#include <stdio.h>

int main( void )
{
        //      二次元配列の初期代入

        int     mark[8][3] = {
                { 45, 50, 65 }, { 64, 72, 84 },
                { 58, 92, 43 }, { 60, 62, 54 },
                { 51, 33, 45 }, { 76, 88, 65 },
                { 45, 53, 35 }, { 39, 62, 38 } } ;
```

生徒の合計と教科の平均を求めるプログラム

```
        int  sum, i, j ;

        for (i = 0; i <= 7; i++) {
                sum = 0 ;
                for (j = 0; j <= 2; j++) {
                        sum = sum + mark[ i ][ j ];
                }
                printf("生徒 %d の合計 %d ¥n", i, sum );
        }

        for ( i = 0 ; i <= 2 ; i++ ) {
                sum = 0 ;
                for ( j = 0 ; j <= 7 ; j++ ) {
                        sum = sum + mark[ j ][ i ] ;
                }
                printf("教科 %d の平均点: %d¥n", i, sum / 8 );
        }
        return 0 ;
}
```