プログラミング演習 第 8 回
配列 I

2000 年 6 月 1 日

目 次

1 目標 **1**

2 配列とは **1**

2.1 1 次元配列の宣言 1

2.2 2 次元以上の配列の宣言 1

3 素数表の作成 (エラト ステネスの篩 (sieve)) **1**

4 実習 **3**

4.1 例 1:1000 までの素数表を出力するプログラム 3

5 配列と関数および前回までの復習 **4**

5.1 例 2: 内積を用いた角度の計算 4

6 さらにポインタ **5**

7 例 3: 配列とポインタ変数の違い **5**

8 文字列に関する注意 **6**

9 練習問題 **7**

1　目標

- C 言語における配列型変数の宣言の仕方, 使い方を知る.

2　配列とは

今回は変数の配列について詳しくやります.
今まで書いてきたプログラムでは,

```
char nyuryoku[80];
```

の部分が, char 型の変数の配列の宣言になっています.
配列は, 同じ型の変数を複数個並べた変数です. 並べ方は, 1 次元的であると限る必要は無く, 2 次元的, 3 次元的であっても構いません. 数学でいうと, ベクトルや行列 (さらにテンソル) だと考えてください.
このように, 基本データ型からより複雑な別のデータ型を定義する事は, 高級言語では必ずできるようになっています. プログラムに有効なデータ型を定義するのは, 重要な事です.

2.1　1 次元配列の宣言

1 次元配列は, 次の形で宣言します.

配列の型　配列変数名 [配列の大きさ];

例えば上の

```
char nyuryoku[80];
```

は, char 型の変数を 80 個並べた配列の宣言になります. すなわち, コンパイラは 80 文字分の文字を納める記憶領域を確保します. この時, 配列の各成分の値は, nyuryoku[0], nyuryoku[1], ···, nyuryoku[79] で参照します. 数学とは違い添字は, 0 から始まります.

2.2　2 次元以上の配列の宣言

2 次元配列は, 次の形で宣言します.

配列の型 配列変数名 [配列の第 1 成分の大きさ][配列の第 2 成分の大きさ];

例えば, double 型の成分を持つ 4 行 5 列の行列を宣言する場合には,

```
double matrix[4][5];
```

とします. 数学から想像される, double matrix[4, 5] とは異なる事に注意して下さい.
3 次元以上の配列の宣言も同様にできますが, それが必要になる事は少ないので, ここでは省略します.

3　素数表の作成 (エラトステネスの篩 (sieve))

配列を用いたプログラムの例として, エラトステネスの篩法で素数表を作成して見ます. エラトステネスの篩は, 小さい素数から順にその数の倍数を消去していくことにより素数を残していく方法で, 素数表の作成には, とても効率が良い事が知られています.

よく知られている事実として, 素数は無限に存在します. 簡単な証明をあげます：

定理 [ユークリッド 原論]

素数は無限に存在する.

(証明)

今仮に素数が有限個存在するとします. 即ち,

$p_1, p_2, \cdots, p_n$.

但し, $p_1 < p_2 < \cdots < p_n$.

ここで, $n = p_1 \cdot p_2 \cdots p_n + 1$ とおくと, これは, 明らかに $p_n < n$ であり, かつ, 新しい素数となります. (何故か?)

故に, 最初の仮定に矛盾するので, 素数は無限個存在する.

(証明終わり)

さて, エラトステネスの篩というのは, 以下のアルゴリズムで素数以外の整数を削除していきます. 簡単な場合として, 38 までの整数から素数を拾い上げる方法を考えてみましょう.

配列を利用しますので, 0 から数え上げます.

```
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
```

(1) 0 と 1 を素数の候補から外す.

(2) 38 以下の 2 の倍数を素数の候補から外す.

(3) 38 以下の 3 の倍数を素数の候補から外す.

(4) 以下順に, 38 以下の $5, 7, 11, \cdots, 38$ の 2 倍数以上の整数を素数の候補から外す.

(5) 残った整数が素数である.

この場合素数表は,

```
2 3 5 7 11 13 17 19 23 29 31 37
```

となります.

これをプログラムで実行させる場合は, 配列を用いて与えられた整数が素数か否かのシンボルを与えて, 最後に, 素数のシンボルをもつ整数のみを出力すればよろしいです. 即ち, あらかじめ, 全ての整数 i に `isprime[i] = 1` というシンボルを与え, 上の (1), (2), (3) のアルゴリズムを行うに従って, 該当する整数 i のシンボルを`isprime[i] = 0` と変えていき, 最後に, `isprime[i] == 1` の整数 i を出力すればよいことになります.

注意

C では, a と b が等しいことは, $a == b$ とかく.

では, 1000 までの素数表を出力するプログラムを作成しましょう.

4　実習

以下の問題の空欄を埋めよ. ファイル名は creoportno.8-1 とする.

4.1　例 1:1000 までの素数表を出力するプログラム

注意

プログラムの最初の方にある,

```
#define MAX 1000
```

は, マクロ置換というもので, この 1000 の値を変えてコンパイルしなおせば, その値までの素数表が出力されます. (ただし, 配列の添字が取れる範囲の制限が付きます. この値はおそらく処理系依存です.) これを, 1000 とするとプログラムの変更のたびに, 4 箇所の値の変更が必要になり, プログラムのミス (バグ (bug) と言われる) の原因になりますので, このような工夫をします.

また, 配列も他の変数と同様宣言した時の初期値は不定です. これをきちんと初期化しないと, プログラムは動きません.

```
/* Print primes to MAX  */

#include <stdio.h>
#define MAX 1000

main()
{
        char  isprime[MAX];          /* 素数判定の配列  */
        int   i, j;

        for((1))
        {   (2)         }             /* 配列 isprime[i] の初期化 */

        /* 篩のアルゴリズム始まり */
        isprime[0] = isprime[1] = (3);           /* 0 と 1 は素数ではない */
        for (i=2; i < MAX; i++){
                if (isprime[i]==1){              /* i が素数のとき */
                        for ((4)){
                            isprime[i*j] = (5); /* i の 2 倍数以上の倍数を削除 */
                        }
                }
        }                                        /* 篩のアルゴリズムの終了 */

        for (i=0; i <MAX ; i++){                /* 素数の表示 */
                if ((6))
                        printf("%3d\t", i);   /* 3 桁分の幅で出力  */
        }
}
```

5　配列と関数および前回までの復習

関数の宣言の時に述べましたように, 配列は関数の返り値や引数にする事はできません. さらに, 配列全体を代入する事もできません. しかし, 関数間での参照ができないと, プログラミングをする上で, とても不便です. 配列は, 前回述べたポインタを通して, 関数間での参照をします[1] .

例えば,

```
int a[10];
```

と宣言した時, a は配列の先頭要素へのポインタです. すなわち, a と　&a[0] はこの意味で, 全く同じ意味を持ちます. このことを利用して, ポインタ経由で配列を関数にわたす事ができます. 次の例は, このことを利用して内積関数を作り, 3 次元空間内の 2 つのベクトルのなす角を弧度法で出力するプログラムです.

5.1　例 2: 内積を用いた角度の計算

このプログラムで, 関数　inn_prod と言う関数は, double 型のポインタを引数にしていますが, それに double 型の配列変数名を渡しています. もちろん, これは内積を計算する関数です.

また, 最後の角度の計算で acos と言う関数が使われていますが, これは cos の逆関数で, 返り値はラジアン単位です. これは, 数学関数ライブラリに入っています. また, 数学関数ライブラリを使いますから, コンパイル時にはオプション -lm を付けて下さい.

```
/* File name 8-2.c                                    */
/* Calculate the angle of two 3-dimensional vectors */

#include <stdio.h>
#include <math.h>

double inn_prod(double *, double *);

main()
{
        char nyuryoku[80];
        double a[3], b[3];
        double angle;

        printf("最初のベクトルの x, y, z 成分を空白で区切って入力して下さい>>");
        gets(nyuryoku);
        sscanf(nyuryoku, "%lf %lf %lf", &a[0], &a[1], &a[2]);

        printf("次のベクトルの x, y, z 成分を空白で区切って入力して下さい>>");
        gets(nyuryoku);
        sscanf(nyuryoku, "%lf %lf %lf", &b[0], &b[1], &b[2]);

        angle = acos(inn_prod(a, b)/sqrt(inn_prod(a, a)*inn_prod(b, b)));
```

[1] この辺の ANSI の規格は不徹底で, 構造体が関数の引数, 返り値にできたり, 代入ができたりしますので, 配列もこれを利用すれば, 引数返り値にもできますし, 代入もできます. 配列だけを特別扱いするのは, 規格上明らかにおかしな事です

```
        printf("2 つのベクトルのなす角はおよそ %f ラジアンです\n", angle);
}

double inn_prod(double *a, double *b)
{
        return a[0]*b[0]+a[1]*b[1]+a[2]*b[2];
}
```

6 さらにポインタ

この講義で幾度とでて来る言葉ですが, 上の配列との関係でもう少し詳しく述べておきます.

そもそもポインタとは,「場所を指し示すための物」という意味で, C 言語でもこの意味で用います.

前回の講義で出ましたが,

```
int *pa;
```

の宣言は, pa をポインタ型の変数として宣言します. すなわち, pa は整数を指し示すための物です.

今回の配列の宣言

```
int a[10];
```

でも a は配列の先頭へのポインタであると言いました.

この 2 つの違いうちの重要な 1 つが,「変数」か否かです[2] . すなわち, 最初の宣言では, pa は int 型へのポインタ変数で, その値をプログラムの中で, 代入やインクリメント (1 増やす), デクリメント (1 減らす) 等の操作で変化させる事ができます. これに反して, 2 番目の宣言では, a は変数ではありません. すなわち, a に値を代入したり, a の値を変化させようとすると, コンパイラエラーになります.

すなわち,「ポインタ変数」と一般的な「ポインタ」と言う言葉は, 集合の包含関係にあることを, 注意してください.

7 例 3: 配列とポインタ変数の違い

上に述べた違いを, 次の 3 つで実験してください. 最初は, きちんと動くプログラムです. putchar は文字を出力するライブラリ関数です. ポインタや配列は定義と同時に初期値を与えていますが, これらの解説は, 後でします.

プログラムの中で, ポインタ変数の値を変化させている事に注意してください. この変化がどのような影響を及ぼすかは, 後で述べるかも知れませんが, 詳しくは何らかの本を読んでください.

```
/* File name 8-3.c */
#include <stdio.h>

main()
{
        char *p="This is a test.\n";

        while(*p){
```

[2] 他にも違いがありますが, それは今後の講義で触れるかも知れません

```
                putchar(*p);
                p++;
        }
}
```

しかし, p を配列型で宣言した次のプログラムは, コンパイル時にエラーになります. p++ と p の値を変化させようとした事に問題があります.

```
/* File name 8-4.c */

#include <stdio.h>

main()
{
        char p[]="This is a test.\n";

        while(*p){
                putchar(*p);
                p++;
        }
}
```

しかし, 良く似ていますが次のプログラムはきちんと動きます. ここで *(p+i) という部分では, p+i が指す場所にある内容を取ると言う意味で, p の値を変化させていないからです.

```
#include <stdio.h>

main()
{
        char p[]="This is a test.\n";
        int i=0;

        while(*(p+i)){
                putchar(*(p+i));
                i++;
        }
}
```

8 文字列に関する注意

C 言語では, 文字列型と言う変数型が存在しません. 文字列は, 全て char 型の配列であり[3] , 文字列の最後には必ず数としての 0, C 言語の文字記号で '\0' が付きます (null terminated string と言う). また, 文字列の初期化ではダブルクォーテーション " で囲む事により, 最後の '\0' が自動的付加されます.

上のプログラムで,

[3] おそらくこのことが, 関数は配列を引数にできないとか, 関数は配列を返せないとかの規格決定に影響した物だと思いますが, 真実は知りません

```
char *p="This is a test.";
```

となっている所は, 次のように読みます.

- p は, char 型を指すポインタ型変数である.
- p は, 文字列型配列 "This is a test." の最初の文字を指すように, 初期化する.

また,

```
char p[]="This is a test."
```

は, 次のように読みます.

- p[] は, 続く文字列が納めるだけの大きさを持った配列である.
- p[] は, 続く文字列を初期値として持つ.

いずれの場合にも, 文字配列の大きさは, 文字数 + 1 でこの場合だと 16 です. +1 の部分は, 終端文字 '\0' の部分です.

また, 文字列が必ず数の 0 で終る事を利用しる事により, 次の while 文が問題無く実行されているのです. すなわち, 0 以外はループで回し, 終端の 0 が来た時点でループは終了します.

9 練習問題

(1) (レポート問題) 正の整数の素因数分解を求めるプログラムを作成せよ.

例えば, $6 = 2 * 3$ と表示されるようにする.

(2) 3 次元 $\mathbf{R}^3$ の 2 つのベクトルを与えられたとき, それに直交するベクトルを求めるプログラムを作成せよ. なお, 平行なベクトルが与えられた時には, そのことを告げるようにする事.

Subject 名はそれぞれ creportno.8-1, creportno.8-2 とし, 締め切りは 6 月 13 日とします.
但し, (2) は希望者だけ.