第5回
プログラミング及び実習 II

第4回課題の回答例

ex04-1：実数の整数部と小数部

```
#include <stdio.h>

int main() {
    double x;
    int n, p;

    scanf("%lf", &x);
    n = x;
    p = (x-n)*10000;
    printf("%d %d\n", n, p);

    return 0;
}
```


ex04-2：文字整数型

```
#include <stdio.h>

int main() {
    int a;

    while (1) {
        scanf("%d", &a);
        printf("%d", a);
        if (a < 33 || 126 < a)              if (33 <= a <= 126)
            break;
        else
            printf(" %c\n", a);
    }
    printf("\n");
    return 0;
}
```

ex04-3：繰り返し処理

```
#include <stdio.h>

int main() {
    int i, n;
    int sum = 0;
    scanf("%d", &n);

    for (i=1; i<=n; i++) {
        sum += i;
        printf("%d\n", sum);
    }
    return 0;
}
```

：

ex04-4：三角関数と繰り返し処理の中断

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

int main() {
    int a;

    do {
        scanf("%d", &a);
        if (abs(a) >= 360)
            continue;

        double rad = a*M_PI/180;
        printf("%f %f\n", sin(rad), cos(rad));
    } while (a != 0);

    return 0;
}
```


ex04-5：対数関数

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

int main() {
    double a, b;

    do {
        scanf("%lf%lf", &a, &b);
        if (a == 1.0)
            break;
        else if (a <= 0 || b <= 0)
            continue;

        printf("%.8f\n", log(b)/log(a));
    } while (1);

    return 0;
}
```


ex04-6：3つの整数の比較

```
#include <stdio.h>

int main() {
  int a, b, c;

  while (1) {
    scanf("%d%d%d", &a, &b, &c);

    if ( a == b && b == c) {
      if (a == 0) break;
      printf("すべて同値\n");
    }
    else if (a == b || b == c || a == c)
      printf("2つが同値で1つ異なる\n");
    else
      printf("すべて異なる\n");
  }

  return 0;
}
```

ex04-7：マクローリン展開

```
#include <stdio.h>

int main() {
  int m;
  double x;
  scanf("%d%lf", &m, &x);

  int k, fac;
  double pow, sum;

  for (k=0; k<=m; k++) {
    if (k==0)
      fac = pow = sum = 1.0;
    else {
      fac *= k;
      pow *= x;
      sum += pow/fac;
    }
    printf("%d %.10f\n", k, sum);
  }

  return 0;
}
```


ex04-8：n次元ユークリッド球面の体積

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

int main() {
  int n, k;
  double r;

  scanf("%d%lf", &n, &r);
  if (n <= 0 || r <= 0) exit(1);

  k = n/2;

  double fact_k=1;
  int i;
  for (i=0; i<k; i++) {
    fact_k *= i+1;
  }

  double v;
  if (n%2 == 0) {
    v = pow(M_PI, k)/fact_k * pow(r, 2*k);
  } else {
    double fact_k2=fact_k;
    for(i=k; i<2*k+1; i++) {
      fact_k2 *= i+1;
    }
    v = 2*fact_k*pow(4*M_PI,k) / fact_k2 * pow(r, 2*k+1);
  }

  printf("%f\n", v);
  return 0;
}
```


本日の内容

- #define 定数マクロ定義
- 配列変数
  - 配列変数の宣言
  - 配列変数の参照・代入
  - 配列変数の初期化

#define 定数マクロ定義

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

#define PI 3.14159265358979323846

int main() {
      :
```

プログラム上の PI はコンパイル前に全て 3.1415926... という文字に置き換わる

じつはπの値は /usr/include/math.h に
#define M_PI 3.14159265358979323846
されています

意味や役割が分かりづらい数値などの定数を、任意の文字列で置き換えることができます。

あとで変更するかもしれない定数なども、#define で定数として定義しておくと、プログラムソースのあちらこちらに散らばった数値をまとめて変更できて便利です。

#define 定数マクロ定義

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

#define PI 3.14159265358979323846

int main() {                          double rad = x/180 * 3.14159265358979323846;
     :
double rad = x/180 * PI;
    printf("sin(%f) = %f\n", x, sin(rad));
    printf("cos(%f) = %f\n", x, cos(x/180*PI));
     :
```

!!

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

#define PI 1+2.14159265358979323846

int main() {                          double rad = x/180 * 1+2.14159265358979323846;
     :
double rad = x/180 * PI;
    printf("sin(%f) = %f\n", x, sin(rad));
    printf("cos(%f) = %f\n", x, cos(x/180*PI));
     :
```


本日の内容

#define 定数マクロ定義

配列変数

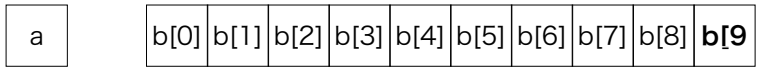
- 配列変数の宣言
- 配列変数の参照・代入
- 配列変数の初期化

配列変数とは

```
int main() {

  int a;          a という名前の整数型の変数を1つ用意

  int b[10];      b[0], b[1], … , b[9] という名前の整数型の
     :            変数を合計10こ用意
```



b という整数型の変数は存在しない。b[0] や b[9] が一つ一つの変数。

b[n] と宣言すると、b[0] から b[n-1] までが使える。b[n] は使えない。

配列変数の [ ] の中の何番目かを表す0以上の整数を「添字」と呼ぶ。

配列変数の何がうれしいのか

例えば、**1000個の変数**が必要なときに、

```
int a0, a1, a2, a3, ..., a998, a999;
```

としなくても、

```
int a[1000];
```

こうして宣言できる。
また、for 文などを使って、こんなふうに使える。

```
for (i=0; i<1000; i++) {
    a[i] = i;
}
```

配列変数が無ければ、a0=0; a1=1; a2=2; a3=3; … a999=999;

配列変数の参照・代入

添字が付く以外は、通常の変数と同じ

```
int a[10];

a[0] = 0;
a[1] = 1;
a[2] = a[0] + a[1];
a[10] = 99;   // 宣言外の添字はエラー
```

ただし、添字を整数で表現できるので

```
double x[10];
int i = 0;

x[i] = 0.1;              // x[0] = 0.1
i++;                     // i = 1
x[i] = 0.2;              // x[1] = 0.2
x[i*3] = x[i] * x[0];    // x[3] = x[1] * x[0]
```


変数での配列変数宣言：C99規格

配列のサイズに変数を利用することができます。

```
int n = 10;
int a[n];        // int a[10]
```

したがって、動的に（実行時に）サイズを指定できる

```
int n;
scanf("%d", &n);
if (n < 0) exit(1);

int a[n];
```


例：配列変数への入力・出力

整数型の配列変数 a を大きさ(要素の数)10で宣言

a[0] から a[9] まで、scanf() で整数を入力

入力した a[] をすべて順番に出力

```
int a[10];

for (i=0; i<10; i++)
  scanf("%d", &a[i]);

for (i=0; i<10; i++)
  printf("a[%d]=%d\n", i, a[i]);
```


配列変数の初期化

基本的には全ての要素に逐一代入する

```
for (i=0; i<10; i++) {
    a[i] = i*2;
}
```

宣言時に定数で初期化することもできる

```
int a[10] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
```

配列変数の初期化

宣言時に初期化することもできる...が、

```
int a[10] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
```

もし、要素数より多くの値で初期化すると

```
int a[10] = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
```

コンパイル時に警告(Warning)される(エラーではない)

もし、要素数より少ない値で初期化すると

```
int a[10] = {0, 2, 4, 6};
```

以降の要素は全て 0 で初期化される。

宣言時の要素数を省略すると、初期化値の数でサイズが決まる

```
int a[] = {0, 2, 4, 6, 8};
// int a[5] = {0, 2, 4, 6, 8};
```


例：配列変数で総和を計算

大きさ(要素の数)が10の実数型の配列変数 x に対して、

各要素 x[0] から x[9] までの値を順にキーボードから入力すると、

x[0] から x[9] の値とそれらの総和を出力する

```
double x[10];
double sum = 0.0;

int i;
for (i=0; i<10; i++) {
  scanf("%lf", &x[i]);
  sum += x[i];
}

for (i=0; i<10; i++)
  printf("x[%d] = %f\n", i, x[i]);

printf("sum = %f\n", sum);
```