ソフトゼミA　第６回 解答と解説

※今回の解説は、**main 文**およびプリプロセッサ命令(**#include ～**)を**省略**しています。

■ 練習問題

1. 3 ある数の絶対値を返す関数 double get_abs(double x)を作れ。

```
double get_abs( double x ){
  if( x >= 0 ){
    return x;
  }else{
    return -x;
  }
}
```

絶対値の定義

$$|a| = \begin{cases} a\ (a \geq 0) \\ -a\ (a < 0) \end{cases}$$

に基づいて組むだけです。なお、条件演算子というものを使って、

```
double get_abs( double x ){
  return ( x >= 0 ) ? x : -x;
}
```

と書くこともできます。

2. ある正の整数 a から b までの n の倍数を全て表示する関数 void display_numbers(int a, int b, int n)を作れ。

```
void display_numbers( int a, int b, int n ){
  int i;
  for( i = a; i <= b; i ++ ){
    if( i % n == 0 ){ printf( "%d¥n", i ); }
  }
}
```

何度も解説していますが、n の倍数であるということは n で割った余りが 0 であることと同じです。この例では、数字を 1 行に 1 つずつ書いていますが、横にズラズラ並べても構いません。

3. a06_2.c を改造して、半径 r の球の表面積を求める関数 double sphere_s(double r) を作れ。ただし、球の表面積 $S = 4\pi r^2$ である。

```
double pi( void ){
  return 3.141592654;
}
double sphere_s( double r ){
  return 4 * pi() * r * r;
}
```

本編でも書きましたが、pi はグローバル変数でも OK ですが、a06_2.c の改造ということでそのままにしてあります。4 * pi() * r * r の計算結果をどこかの変数にとっておいて、それを return しても OK ですが、この例では式の計算結果を直接 return しています。

■ 追加練習問題

1. 配列引数を用いて、フィボナッチ数列の各項を配列の各要素に格納する関数を作れ。オーバーフローする手前の項まで求めること。ただし、配列は 100 要素分確保すること。

```
int fibo( int a[]  ){
  int i;
  int temp;
  for( i = 0; i < 100; i++ ){
    if( i <= 1 ){ a[ i ] = 1; }
    else{
      temp = a[ i - 2 ] + a[ i - 1 ];
      if( temp < 0 ){ return i; } /* Overflow! */
      a[ i ] = temp;
    }
  }
  return 100;
}
```

漸化式の通りに組むとこのようになります。オーバーフローした時には、オーバーフローする直前までの要素の個数を返します。(これによって、main 文の方で何要素出力すればよいかをわかるようにしています。) オーバーフローの判定は、**その数がマイナスになった時**としています。オーバーフローについては∀第 2 回の資料を参照のこと。

2. 配列引数で受け取った int 型配列 a の a[0]～a[n - 1]を、昇順に並び替える関数 sort(int a[], int n)をつくれ。

```
void sort( int a[], int n ){
  int i, temp, ng;
  do{
    ng = 0;
    for( i = 1; i < n; i++ ){
      if( a[ i - 1 ] > a[ i ] ){
          ng = 1;
          temp = a[ i - 1 ];
          a[ i - 1 ] = a[ i ];
          a[ i ] = temp;
      }
    }
  }while( ng );
}
```

並び替え(ソート)の方法はいくつかありますが、ここでは最も簡単な方法の一つである方法の一つである「バブルソート」で解説します。

例えば、a[0]から a[4]まで順に 2, 7, 1, 8, 2 な配列を受け取ったとします。

2	7	1	8	2

ここで、a[0]と a[1]を比較します。

2	7	1	8	2

これは昇順に並んでいるのでスルーします。

次に、a[1]と a[2]を比較します。

2	7	1	8	2

これは昇順ではないので、入れ替えます。入れ替えると、次のようになります。

2	1	7	8	2

次に、a[2]と a[3]を比較します。

2	1	7	8	2

スルー。

最後に、a[3]と a[4]を比較します。

2	1	7	8	2

↓

2	1	7	2	8

入れ替え。

という動作を、数字が完全に昇順に並ぶまで繰り返します。入れ替えが 1 度も行われない時に、ループを抜けます。

ソートのアルゴリズムには様々な種類があります。その中でもこの方法はかなり遅い部類に入ります。特に、データ件数が増えれば増えるほど苦しくなってきます。関数の再帰呼び出しを用いると、素早く(なることがある)「クイックソート」という方法をとることができます。興味があれば是非とも調べてみましょう。