担当：　富井尚志　(tommy@ynu.ac.jp)

第 4 回：配列を扱うアルゴリズム（1）

☆ 関数の呼出し（前回の復習と確認）

```
int main(void)
{
  int a = 10;

  printf("%d¥n", f(a));
  return 0;
}
```

```
int f( int x )
{
    x++;
    return x;
}
```

・**引数**

関数に対して入力として指定するパラメータ
関数の括弧の中に書く

・**実引数と仮引数**

実引数：関数呼出しの部分で関数の値を求めるために実際に値を与える引数.
仮引数：関数の定義の部分で入力となる変数．呼出しのたびに新たに生成され(**自動変数**)，実引数の値が代入される.

・**プログラム内での関数の振舞**

「**関数呼出**」により対応する関数の内部の処理に移行する.「関数呼出」は関数に実引数を与え，その関数の値を求めることである（右上の f(a) 自身の値を求めること）.プログラムを実行している時に，関数を含む式たとえば f(a) に遭遇したとすると，その式の値を求めるために以下のことが行なわれる.

1) **全ての(実)引数の値が求められる**．f(a) の場合なら，a の値を求める．つまり，10.
2) 1)で求めた値を，関数定義の中の対応する**仮引数の変数に代入**する．上の例では，変数 x に 10 が代入される.
3) **関数内部の文が実行**される.上の例では,x++ (x の値に 1 を加えた値を再び x に代入), return x が順番に実行される.
4) return 文か関数の最後の文が実行されると，この関数の実行は終了．**return 文で終ればその後ろの式の値が関数呼出しの結果の値となる**．(return 文がなければその関数は値を持たない関数だとみなされる．) 上の例では，return x によりこの時点の x の値 11 が関数呼出 f(a) の値となる.
5) この値が**関数の値**である．つまり，f(a) を関数呼出しの値 11 で置き換える，と考えれば良い.

☆ 配　列　（再出）

◎ 配列って?

・データの構造化手法の一つ
・同じ型の複数のデータを**添え字**(インデクス)により参照できる変数「**同じ型の別々のデータが並んでいる**」
・添え字 → 整数式なので計算により決めることができる.
・様々な応用が考えられる.

◎ 宣言の方法

n[1]からでなく，n[0]からである点に注意！

double n[10]; → n[0],n[1],...,n[9]という 10 個の倍精度浮動小数点型変数を用意したのと同じ.

※注意：宣言の数字は添字の上限ではない．用意するデータの　　である.

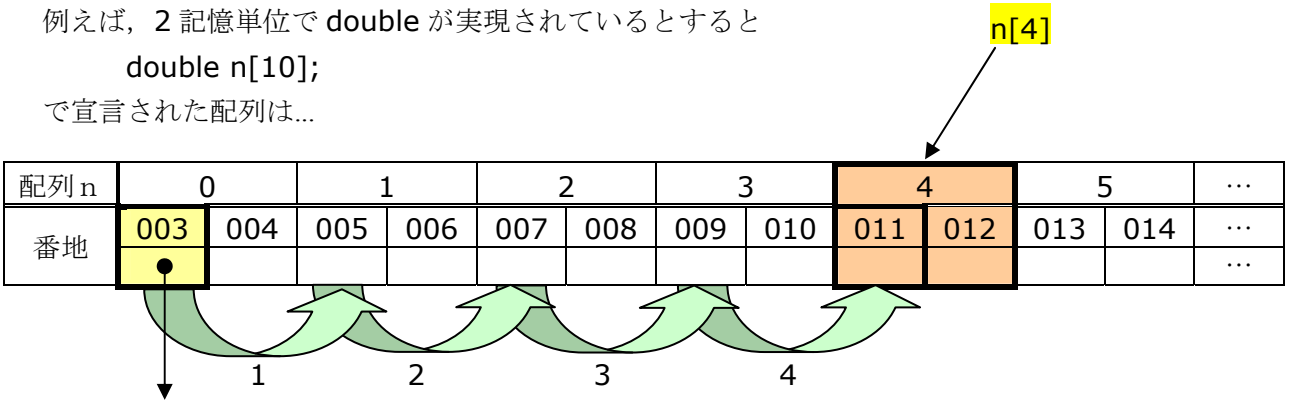
この印の部分は講義中に書き入れてください

◎ 計算機内部の記憶と配列の関係

例えば，2 記憶単位で double が実現されているとすると

double n[10];

で宣言された配列は…



この例では，3 番地から順番に double で必要な記憶（配列の大きさ）が確保されている．

- 記憶のどの部分から配置されるかは［　　］が決定する（プログラマが気にする必要はない）．
- 配列の添字から実際のデータの番地を計算してその値を取り出してくれる（ようにコンパイラが，利用者が書いたソースプログラムから実行プログラムを作成してくれる）．
- 配列のデータは内部では「**先頭番地** ＋ **添字**」によって指定される．

［　　］はその配列の先頭番地を示す．

※ 実は，n[...] における n は先頭番地を表している．

n[4] → 番地（n + double 4 コマ）の中身（double 1 コマ分をデータとして）

→ 番地(n + 8)の中身 → 上の例では 番地(3 + 8 (= 11))の中身

（double 1 コマ分をデータとしてみるため，番地 11 と 12 の中身をつなげて double としてみなす）

重要

◎ 多次元配列

・宣言方法

int k[2][3];

k[0][0], k[0][1], k[0][2], k[1][0], k[1][1], k[1][2]
という 2×3＝6 個の整数型変数を用意したのと同じ．
（2 次元配列）

データの順序に注意せよ：
まず第 1 要素を 0 にして第 2 要素を動かし，次に第 1 要素を 1 にして第 2 要素を動かし…（以下同様）となる．

k[i][j]		j		
		0	1	2
i	0	4	2	1
	1	7	10	3

◎ 配列の初期化

```
int n[10] = { 2, 4, 10, 3, 5, 2, 7, 1, 21, 100 };
int n[] = { 2, 4, 10, 3, 5, 2, 7, 1, 21, 100 };
int k[2][3] = { { 4, 2, 1 }, { 7, 10, 3 } };
char name1[] = { 't', 'a', 'r', 'o', '¥0' };
char name2[] = "taro";
```

```
k[0][0] = 4;
k[0][1] = 2;
k[0][2] = 1;
k[1][0] = 7;
k[1][1] = 10;
k[1][2] = 3;
```
としたのと同じ．

文字列の最後はヌル記号 '¥0'.
文字列で宣言したときは自動的に最後にヌル記号が入る．

◎ 文字列

文字列は文字配列

☆配列を用いたプログラム例

◎ 線形探索

・探索

既に登録されているデータの中に，いま，調べたいデータが入っているか？

- 入っている場合　→ 見つかったことを示す（そのデータの内容提示）
- 入っていない場合 → 無いことを示す

・線形探索

はじから順番に 1 つずつ調べていく．

・通常のプログラム（ → lsearch1.c（本資料 4 頁））

・番人（sentinel）ありのプログラム（ → lsearch2.c（本資料 5 頁））

配列の最後に見つけるべきデータを予め配置する．繰り返しの終了条件が簡単化される．

◎ 2 分探索

・データが昇順あるいは降順で整列している時に使える．（ → bsearch.c（本資料 6 頁））

・ある区間のちょうど真中に位置するデータを調べる．

- 同じなら見つかった
- 探したいデータの方が　大きいなら　上半分を同様に探索
- 小さいなら　下半分を同様に探索

・探索の区間を繰り返し半分にするので，高速 → オーダ O(log n) の操作手順（n はデータ数）

◎ 最大値（最小値）

・1 番目～(i-1)番目までの最大値がわかっている(max)とすると，1 番目～i 番目までのデータの中の最大値は，

max >= （i 番目のデータ） なら max

max < （i 番目のデータ） なら（i 番目のデータ）

となる．求まった最大値を 改めて max とする．

・上記の考え方を，i = 2 から n まで変化させて適用する．（ → max.c（本資料 7 頁））

今日のポイントは上の 3 つのアルゴリズムです．比較的簡単ですが，重要ですのでしっかり覚えて下さい．

```
/*****************************************************************
      アルゴリズムとデータ構造
      サンプルプログラム  lsearch1.c
      <<線形探索>>
      copyright (c) 1995,96,97    T.Mori <mori@forest.dnj.ynu.ac.jp>
*****************************************************************/
#include <stdio.h>

#define N 10    /* 記号定数の定義．N は 10 に置き換えられる */
int a[N] = { 3, 54, 16, 8, 9, 1, 5, 22, 19, 60 };

void search( int x );

int main(void)
{
   search( 3 );
   search( 5 );
   search( 30 );
   search( 60 );
   return 0;
}

void search( int x )
{
   int i = 0;

   while(   i != N   &&   a[i] != x   )
      i ++;

   if ( i != N )
      printf("%d:  みつかりました¥n", x);
   else
      printf("%d:  みつかりませんでした¥n", x);
}
```

コメント

[　　]が定義され，初期値が代入される

関数 search のプロトタイプ宣言

色々な値で search を実行

「i≠N　かつ　a[i]≠x　の間，i=i+1; を実行しなさい」ということは，
　i=N になったら　　または　　a[i]=x になったら
この処理は終了する．

i≠N なら，
　x の値と“みつかりました”を表示し，
i=N(=10)なら(←全部調べたがなかった)
　x の値と“みつかりませんでした”を表示．

【実行結果】

[　　]

[　　]

[　　]

[　　]

自分で考えて，書き入れて下さい．

```
/*************************************************************
      アルゴリズムとデータ構造
      サンプルプログラム  lsearch2.c
      <<線形探索  番人版>>
      copyright (c) 1995,96,97    T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>

#define N 10    /*  記号定数の定義．N は 10 に置き換えられる  */
int a[N+1] = { 3, 54, 16, 8, 9, 1, 5, 22, 19, 60 };     /*  [          ]用に 1 つ配列要素を多くとる  */

void search( int x );

int main(void)
{
   search(3);
   search(5);
   search(30);
   search(60);
   return 0;
}

void search( int x )
{
   int i = 0;

   a[N] = x; /*  番人の設定  */

   while(    a[i] != x    )
      i ++;

   if ( i != N )
      printf("%d:  みつかりました¥n", x);
   else
      printf("%d:  みつかりませんでした¥n", x);
}
```

a[0]～a[9]までが定義され，初期値が代入される．
a[10]は定義されただけで，値は代入されていない．

関数を呼び出す側から代入された引数 x の値を，万人用の配列要素 a[N](=a[10])に代入している．

「a[i]≠x の間，i=i+1; を実行せよ」
繰り返しの条件が lsearch1.c より簡略化されている
ことがわかる．

【実行結果】

```
[                                   ]
[                                   ]
[                                   ]
[                                   ]
```

自分で考えて，書き入れて下さい．

```
/*****************************************************************
      アルゴリズムとデータ構造　サンプルプログラム bsearch.c
      <<2 分探索>>　copyright (c) 1995,96,97　T.Mori <mori@forest.dnj.ynu.ac.jp>
*****************************************************************/
#include <stdio.h>
#define N 10
int a[N] = { 1, 3, 5, 8, 9, 16, 19, 22, 54, 60 };
/* 2 分探索では各要素が                    ことが必要 */

void search( int x );

int main(void)
{
   search(3);
   search(5);
   search(30);
   search(60);
}

void
search( int x )
{
   int   i, j, k;

   i = 0;
   j = N - 1;

   do {
      k = ( i + j ) / 2;
      if ( x > a[k] )
         i = k + 1;
      else
         j = k - 1;
   } while ( a[k] != x   &&   i <= j );

   if (a[k] == x)
      printf("%d: みつかりました¥n", x);
   else
      printf("%d: みつかりませんでした¥n", x);
}
```

j = 9;

x = 19（=a[6]）のときの変数の値の変化の様子

a[k]	0	1	2	3	4	5	6	7	8	9
—	i									j
9	i				k					j
9						i				j
22						i		k		j
22						i	j	k		
16						i, k	j			
16						k	i,j			
19							i,j,k			

時間の流れ

※ 変数 i と j で挟み込むようにして k の値を絞り込んでいる.

「a[k]≠x かつ i≦j の間，上の{　}を実行しなさい」ということは，

a[k]=x になったら　または　i>j になったら

この do～while ループは終了する.

【実行結果】

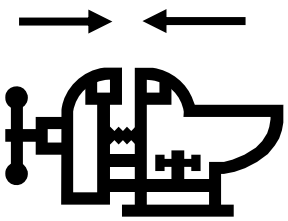
42

43

44

45

自分で考えて，書き入れて下さい.



```
/*************************************************************
      サンプルプログラム  max.c
      <<最大値>>
      copyright (c) 1995    T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>

#define N 10    /*  記号定数の定義．N は 10 に置き換えられる  */
int a[N] = {3, 54, 16, 8, 9, 1, 5, 60, 22, 19};

int main(void)
{
   int i, max;

   max = a[0];

   for ( i = 1;    i < N;    i ++ )
      if ( max < a[i] )
         max = a[i];

   printf("最大値は  %d  です¥n", max);
   return 0;
}

【実行結果】
```

（違っていると思うが）とりあえず，a[0]を最大値とおく．

i の値を 1 から N-1 まで 1 つずつ変えながら a[i] と暫定的な max を比較し，a[i]の方が大きい(より良い解)ならその値を max にする（max = a[i] ）．

最大値を求める常套手段．必ず覚えておくこと．
（最小値も同様に求めることができる．）

自分で考えて，書き入れて下さい．

第 3 回「変数のスコープルール・関数」のまとめ（復習）

- ブロック先頭で宣言される名前 →「**内部変数**」：ここで定義される変数
- **関数自身**：ソースファイル内で宣言された点からそのファイルの終わりまで
- **外部変数**：どの関数にも属さない共通の変数．

◎ 自動変数(内部変数)と外部変数の違い　　～「時間軸」で見た有効範囲の差 ～

- **自動変数**：関数の始め（ブロックの始め）で定義される変数．通常その関数（ブロック）に制御が移るたびに生成され，関数（ブロック）を終了すると，「消滅」
- **外部変数**：すべての関数の外で定義される変数．一回だけ生成され，「常に存在」

◎ 宣言と定義

- 外部的な名前（変数や関数）に関しては区別されるが，内部変数では定義と宣言が同一

☆ 再帰呼び出し

- ある手続き／関数の中から自分自身を呼び出す．
- 自動変数があるので可能．呼出のたびに別の領域が変数に割り当てられる．
- 数学的帰納法などにより解を求める場合などに使う

◎ 前回の出席票の小テストの解答

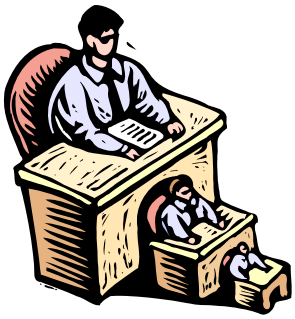
練習問題：　関数の再帰呼び出しを用いて指定した回数だけ画面に"hello, world"と書くプログラムを，下の不完全なプログラム中の［　　］内に適切な記述を補足することによって作成せよ．

```
#include<stdio.h>

void display( int n );
/* 画面に文字を書く関数（プロトタイプ宣言部）  */

int main(void)
{
    int num;

    printf("何回書きますか :" );
    scanf ("%d",&num);                    /* num をキーボードから入力 */
    display( num );
    return 0;
}

/* 関数 display の本体 */
void display( int n )
{
    printf( "hello, world¥n");
    if ( n > 1 ) {
        n--;
        display( n );
    }
}
```

最後にセミコロン；が必要であることに注意せよ．

void は，値を返さない関数に付ける予約語である．なお予約語とは，プログラマが別の意味に使ってはならない単語のこと．

display は関数の名前です．私が勝手に付けたもので，特に深い意味はありません．



◎ 前回の配布資料の演習問題の解答

1－1　2つの引数 n1，n2（ともに int 型）を与えると，それらの値の差（正の数）を返す関数 dif を絶対値関数 abs は使わないで作りなさい（ヒント：なし）

【解答例 1】

```
int dif( int n1, int n2 )
{
    if ( n2 >= n1 )
        return (n2 – n1);
    else
        return (n1 – n2);
}
```

【解答例 2】

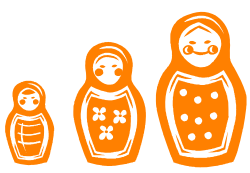
```
int dif( int n1, int n2 )
{
    int n;

    n = n2 – n1;
    if ( n >= 0 ) return n;
    else return –n;
}
```

1－2　任意の正の整数 n（int 型）を与えると，フィボナッチ数 F_n を返す関数 fibo を再帰を用いて作りなさい．

（ヒント：フィボナッチ数 F_n とは，フィボナッチ数列を作る数であり，n=1 または n=2 のとき F_n=1，n が 3 以上のとき $F_n = F_{n-1} + F_{n-2}$ で定義される数である）

【解答例】

```
int fibo( int n )
{
    if ( n <= 2 )
        return 1;
    else
        return fibo( n-1 ) + fibo( n-2 );
}
```



１－３　1-2 のプログラムを使って，n=1 から n=10 までのフィボナッチ数 F_n を画面に表示するプログラムを作りなさい．（ヒント：なし）

【解答例】

```
#include<stdio.h>

int fibo( int n );

int main(void)
{
    int i;

    for(i=1;i<=10;i++)
        printf("F%d = %d¥n",i,fibo(i));
    return 0;
}

int fibo( int n )
{
    if ( n <= 2 )
        return 1;
    else
        return fibo(n-1) + fibo(n-2);
}
```

【実行結果】

```
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
```

$$F_n = F_{n-1} + F_{n-2}$$

１－４　乱数で 100 以下の正の整数を発生させて，それを当てる“数当てゲーム”を作りなさい．例えば実行時は次のような感じです．

```
number = 40
小さいよ
number = 80
大きいよ
・・・・・
number = 55
当たり！
```

（ヒント：stdlib.h と time.h をインクルードすると，次のようにして，int 型変数 ans に 100 以下の正の整数がランダムに入力することができます．

```
int ans;

srand(time(NULL));
ans=(int)(100.0 *rand()/32768.0);
```

【解答例】

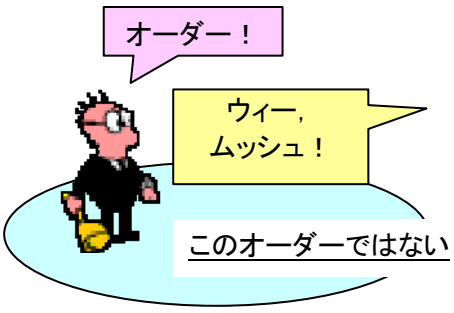
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

int main(void)
{
    int ans,number;

    srand(time(NULL));
    ans = (int)(100.0 * rand()/32768.0);

    do{
        printf("number = ");
        scanf("%d",&number);
        if ( number < ans )
            printf("小さいよ¥n");
        else if ( number > ans )
            printf("大きいよ¥n");
    } while( number != ans );
    printf("当たり！¥n");
    return 0;
}
```

よくある質問とそれに対する解答

1．オーダーって何ですか？

・次に示すオーダー表記があります．

オーダー表記	直観的な意味
$f(n) = o(g(n))$	$f < g$
$f(n) = O(g(n))$	$f <= g$
$f(n) = \Theta(g(n))$	$f == g$
$f(n) = \Omega(g(n))$	$f >= g$
$f(n) = \omega(g(n))$	$f > g$



・例えば，$f(n) = O(g(n))$ とは，任意の正の定数 c に対して，ある n_0 があって，任意の $n > n_0$ に対して $f(n) <= c\ g(n)$ が成り立つことを言います．

・このオーダーはアルゴリズムの良否を比較する際の重要な指針となります．

- **線形探索**：平均で $n / 2$ 回，最悪で n 回の比較が必要．
  線形探索の計算量は $O(n)$（ラージオーエヌ）である
- **２分探索**：およそ $\log_2 n$ 回で探索が終了（証明は略）．
  ２分探索の計算量は $O(\log n)$（ラージオーログエヌ）である

・２分探索の計算時間を $t(n)$ とすると $t(n) = O(\log n)$ である　ことは明らか（なお，底はいくつであっても定数倍しか違わないので，底については論じない）．

・アルゴリズムには，$O(n^2)$，$O(n^3)$，…などのものもあります．

2．２分探索の do～while ループ中で，k = (i + j) / 2; の計算が，i=0，j=9 なら k=4.5 となるのでは？

・この式の右辺は全て整数なので，**整数計算**されます．すなわち，**割り算は切り捨て**となり，値は 4 になります．左辺の k の値も 4 となります．

・もし，k = (i + j) / **2.0**;　となっていれば**実数計算**され，右辺の値は 4.5 になります．この場合は，実数→整数の変換が行われた結果の値（一般には切り捨てて 4）が k の値になります．

・このように，式が出てきたら，**それがどのようにして（整数で？ あるいは 実数で？ どの順序で？）計算されるのかを意識しないと考えるような値にならない**ので注意下さい．

3．リンカって何ですか？

・リンカ（Linker）とは，別々にコンパイルしたオブジェクトファイルを連結して，ひとつのプログラムにするためのソフトウェアです．最近では，リンカを内蔵した統合開発環境が多く，プログラマからリンカの存在が見えにくいです（講義「プログラミング」で出てきますよね？）．

・ちなみに，**電気・電子・情報系の用語では，語尾の長音記号（ー）が省略されることが多い**です
（例：リンカー　→　リンカ）

・それは，３音節以上で最後が er, or などで終わる単語は，最後の長音記号（ー）を付けない習慣があるからです（例外もあります）．一般の人にはやや奇異に感じますが．

- 例 1：プログラマー　→　プログラマ
- 例 2：サーバー　→　サーバ
- 例 3：コンパイラー　→　コンパイラ
- 例 4：インターフェース　→　インタフェース　など．