プログラミング１　第２回
復習（２）

- 変数の種類(前期教科書P189～)
- ２次元配列(前期教科書P228～)
- リニアサーチ

この資料にあるサンプルプログラムは
/home/course/prog1/public_html/2007/HW/lec/sources/
下に置いてあります。**Lec02-***のファイルを自分のディレクトリに
コピーして、コンパイル・実行してみてください

自動変数

```
#include <stdio.h>

void a(void);
void b(void);

main()
{
  int i = 1;

  a();
  b();
}

void a(void)
{
  int i = 2;
  printf("i : %d¥n",i);
}

void b(void)
{
  int i = 3;
  printf("i : %d¥n",i);
}
```

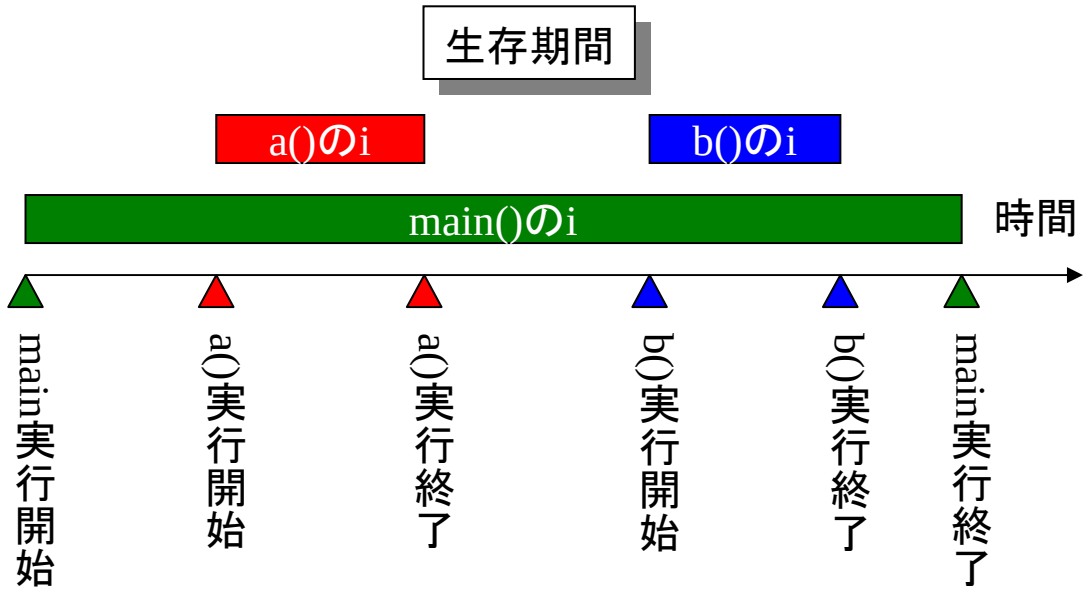
通用範囲

main()のi

a()のi

b()のi

- 関数内のみで通用（通用範囲）
- 関数が呼ばれると生成され、終了すると消える（通用期間）
- 別関数なら（通用範囲が違うので）同じ名前でも構わない



自動変数の実験

```
#include <stdio.h>

void print_ij(void);

main()
{
  int i = 1 , j = 6;

  printf("main : i : %d , j : %d¥n",i,j);
  print_ij();
  printf("main : i : %d , j : %d¥n",i,j);
}

void print_ij(void)
{
  int i = 3 , j = 8;
  printf("func : i : %d , j : %d¥n",i,j);
}
```

左のようなプログラムを作って実験してみると、関数の中のi,jの値はmainの中の値とは異なっていることが分かる。
それに同じ変数名なのにコンパイルエラーにもならない！

実行結果

```
std1ss1{s1000000}1:./a.out
main : i : 1 , j : 6
func : i : 3 , j : 8
main : i : 1 , j : 6
std1ss1{s1000000}2:
```

mainの変数と関数内の変数は同じ名前でも別物である事が分かる

自動変数のまとめ

- **auto（自動）変数**
  - `int i; float a;`などのように他のキーワードを付けないで関数内で宣言すると、自動的に自動（auto）変数になる。
  - 関数の引数も自動的に自動変数になる。
  - 自動変数の「auto」キーワードは実際には全く使用されない
- **register変数**
  - auto変数のうち特に頻繁に処理され、速さが必要なものは
    ```
    register int a;
    ```
    のようにintの前にregisterキーワードをつけ、レジスタ型で宣言する。
- **他に静的、外部変数がある**
  （これらの種類を記憶クラスと呼ぶ）-- 次に説明する

registerとは中央処理演算装置（CPU）内にある非常に速い記憶領域のことである。なお、このごろではコンパイラが優秀で、register宣言をしなくても、非常に良く使用される変数（例えばfor文の制御変数）などは自動的にregisterに割り当てられるので、特にregister宣言をする必要がない場合も多い

外部変数

```
#include <stdio.h>

void a(void);
void b(void);

int x = 0;
```

変数xの通用範囲

```
main(){
  a();
  b();
  a();
  a();

  printf("a/b was called %d times\n",x);
}

void a(void){
  x++;
}

void b(void){
  x++;
}
```

- 全ての関数の外に宣言を置く
- どの関数からも使える（通用範囲が自動変数と異なる）
- プログラム終了まで消滅しない（通用期間も自動変数と異なる）
- 左の例の変数xが外部変数

```
実行結果
std1ss1{s1000000}1: ./a.out
a/b was called 4 times
std1ss1{s1000000}2:
```

静的変数

```
#include <stdio.h>
int a(void);
int b(void);
main(){
  int i,j;

  a(); a(); a(); a();
  b(); b();
  i = a();
  j = b();
  printf("a was called %d times¥n",i);
  printf("b was called %d times¥n",j);
}

int a(void){
  static int x = 0;
  x++;
  return x;
}

int b(void){
  static int x = 0;
  x++;
  return x;
}
```

関数a変数xの通用範囲

関数b変数xの通用範囲

```
実行結果
std1ss1{s1000000}1: ./a.out
a was called 5 times
b was called 3 times
std1ss1{s1000000}2:
```

- 宣言時に「static」というキーワードを付ける
- 関数内に宣言を置き、その関数からのみ使用可能（通用範囲は自動変数と同じ）
- プログラム終了まで消滅しない（通用期間が自動変数と異なる）
- 左の例の関数a/bの変数xがstatic変数
- 変数の初期化は一度目のみ

変数のまとめ（前期教科書P189～）

記憶クラス	宣言場所	キーワード	通用範囲	生存期間	初期化	
					ない場合	ある場合
自動変数	関数内	なし	関数内	関数実行中のみ	初期化されず	毎回
外部変数	関数外	なし	宣言以降全部	最初から最後	0に初期化	１回のみ
静的変数	関数内	static	関数内	最初から最後	0に初期化	１回のみ

- 通用範囲のことをスコープと呼ぶ
- 生存期間のことはデュレーションと呼ぶ場合もある（あまり一般的ではない）
- 関数外で宣言するstatic変数や、extern変数などもあるが、ここでは省略したので、教科書を参照のこと

- （注）自動変数のキーワードは「auto」であるが、実際には使用されていないので、「なし」としてある

記憶クラスの例

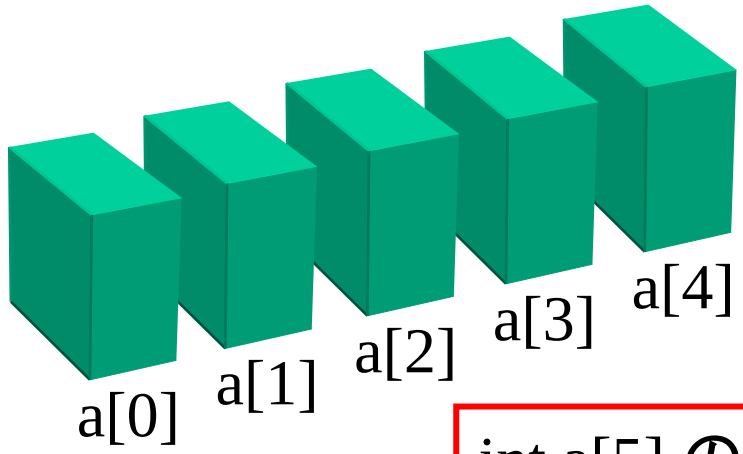
```
#include <stdio.h>
void a(int);

int gbl; /* global variable */
main()
{
  int i; /* automatic variable */

  for(i = 1 ; i <= 10 ; i++){
    a(i);
    printf("%2d : %d\n",i,gbl);
  }
}

void a(int a)
{
  static int x = 0; /* static variable */

  if(x % 2 == 0) gbl = a * a;
  else gbl = a * a * a;
  x++;
}
```

- auto変数：各関数内のi
- 静的変数：関数a内のx
- 外部変数：関数の外のgbl
- この例の関数a()では、偶数回目の呼び出しでは引数の二乗、奇数回目は三乗を行い、外部変数gblに代入している。

実行結果：

```
std1ss1{s1000000}1: ./a.out
 1 : 1
 2 : 8
 3 : 9
 4 : 64
 5 : 25
 6 : 216
 7 : 49
 8 : 512
 9 : 81
10 : 1000
std1ss1{s1000000}2:
```

１次元配列



int a[5] の配列構造

- 添え字が1つの配列を「１次元配列」と呼ぶ。
- これは図にすると上のように、要素が１次元(１列)に並んだようなものだからである

配列の例

- 読み込んだ3つずつのデータの並び方を縦横交換して表示する
- 各行を表示させるのが面倒

実行結果:

```
std1ss1{s1000000}1: ./a.out
1 2 3
2 3 4
3 4 5
4 5 6
^D
  1 |  2 |  3 |  4 |
  2 |  3 |  4 |  5 |
  3 |  4 |  5 |  6 |
std1ss1{s1000000}2:
```

「^D」はControl+Dで、End Of Fileのこと

```
#include <stdio.h>
#define LEN 100
main()
{
  int i, n = 0, x[LEN], y[LEN], z[LEN], status;

  while( 1 ){ /*EOFになるまで無限ループでデータ読み込み*/
    status = scanf("%d%d%d",&x[n], &y[n], &z[n]);
    if(status != 3) break;
    n++;       /* nは入力される行数 */
  }

  for(i = 0 ; i < n ; i++){ /*1行目の表示*/
    printf("%3d |", x[i]);
  }
  printf("\n");

  for(i = 0 ; i < n ; i++){ /*2行目の表示*/
    printf("%3d |", y[i]);
  }
  printf("\n");

  for(i = 0 ; i < n ; i++){ /*3行目の表示*/
    printf("%3d |", z[i]);
  }
  printf("\n");
}
```

scanf()の戻り値について

- scanf()の戻り値は以下のようになっている
  - 通常は成功した読み込みの数
  - EOF(End of File)の場合は−1
- 例のように予定読み込み数（この場合2）ではなかった場合、何か変なデータが入力されたか又はEOFなので、読み込みを終了・中止する処理が必要。

```
#include <stdio.h>
main()
{
  int i,j,status;

  while( 1 ){
    printf("Enter 2 int value : ");
    status = scanf("%d%d",&i,&j);
    printf("i : %d , j : %d , status : %d\n",
            i,j,status);
    if(status != 2) break;
  }
}
```

無限ループ

読み込みが正常でない場合は無限ループ終了

実行結果:

```
std1ss1{s1000000}1: ./a.out
Enter 2 int value : 2 3
i : 2 , j : 3 , status : 2
Enter 2 int value : 1 c
i : 1 , j : 3 , status : 1
% ./a.out
Enter 2 int value : 2 3
i : 2 , j : 3 , status : 2
Enter 2 int value : ^D
i : 2 , j : 3 , status : -1
std1ss1{s1000000}2:
```

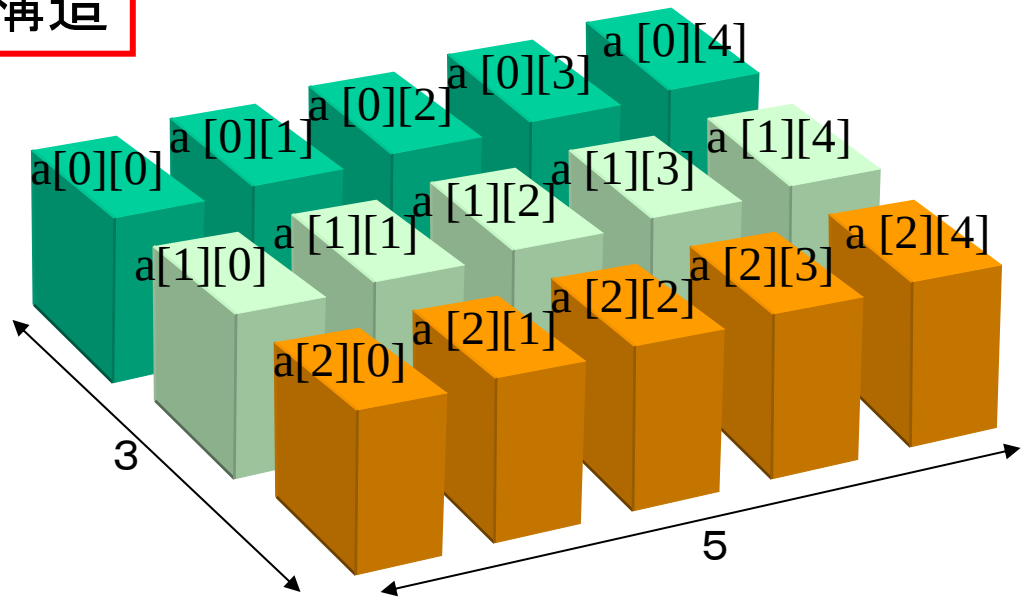
正常

エラー

EOF

１次元配列から２次元配列へ

int a[3][5] の配列構造



- １次元配列の拡張として２次元配列を考える
- これは横に長い（５列）１次元配列を縦に並べた（３行）と思うと分かり易い
- このような３行５列の配列を int a[3][5]のように宣言する

２次元配列の宣言

```
double x[5][10]
```

- この宣言を言葉で言うと次のようになる

変数xは
- x[0][0]からx[4][9]までの50個の要素を持った
- double型の配列で、
- １番目の添字の範囲は0から4、
- ２番目の添字の範囲は0から9までである。

二次元配列の例

- 先の例を2次元配列に変えてみた
- 表示部分で二重ループが使用出来、見やすくなった

```
実行結果:
std1ss1{s1000000}1: ./a.out
1 2 3
2 3 4
3 4 5
4 5 6
^D
  1 |  2 |  3 |  4 |
  2 |  3 |  4 |  5 |
  3 |  4 |  5 |  6 |
std1ss1{s1000000}2:
```

```
#include <stdio.h>
#define LEN 100

main()
{
  int i, j, n = 0, data[LEN][3], status;

  while(1){ /*EOFまで無限ループでデータ読み込み*/
    status = scanf("%d%d%d", &data[n][0],
                   &data[n][1], &data[n][2]);
    if(status != 3) break;
    n++;
  }

  /*2重ループによる2次元配列の表示(縦横交換)*/
  for(j = 0 ; j < 3 ; j++){
    for(i = 0 ; i < n ; i++){
      printf("%3d |", data[i][j]);
    }
    printf("\n");
  }
}
```

```
#include <stdio.h>
#define LEN 100

main()
{
  int i, j, n = 0, data[LEN][3], status;

  while(1){ /*EOFまで無限ループでデータ読み込み*/
    status = scanf("%d%d%d", data[n][0],
            data[n][1], data[n][2]);
    if(status != 3) break;
    n++;
  }

  /*2重ループによる2次元配列の表示（縦横交換）*/
  for(j = 0 ; j < 3 ; j++){
    for(i = 0 ; i < LEN ; i++){
      printf("%3d |", data[i][j]);
    }
    printf("＼n");
  }
}
```

Quiz: 左のプログラムはどこがへんですか？

二次元配列の初期化

- ２次元配列の初期化は１次元配列の初期化を複数個書くことで行う。（{}はどれだけ入れ子にしても構わない）

```
int a[3][5] = {
    {1,2,3,4,5},
    {6,7,8,9,8},
    {7,6,5,4,3},
};
```

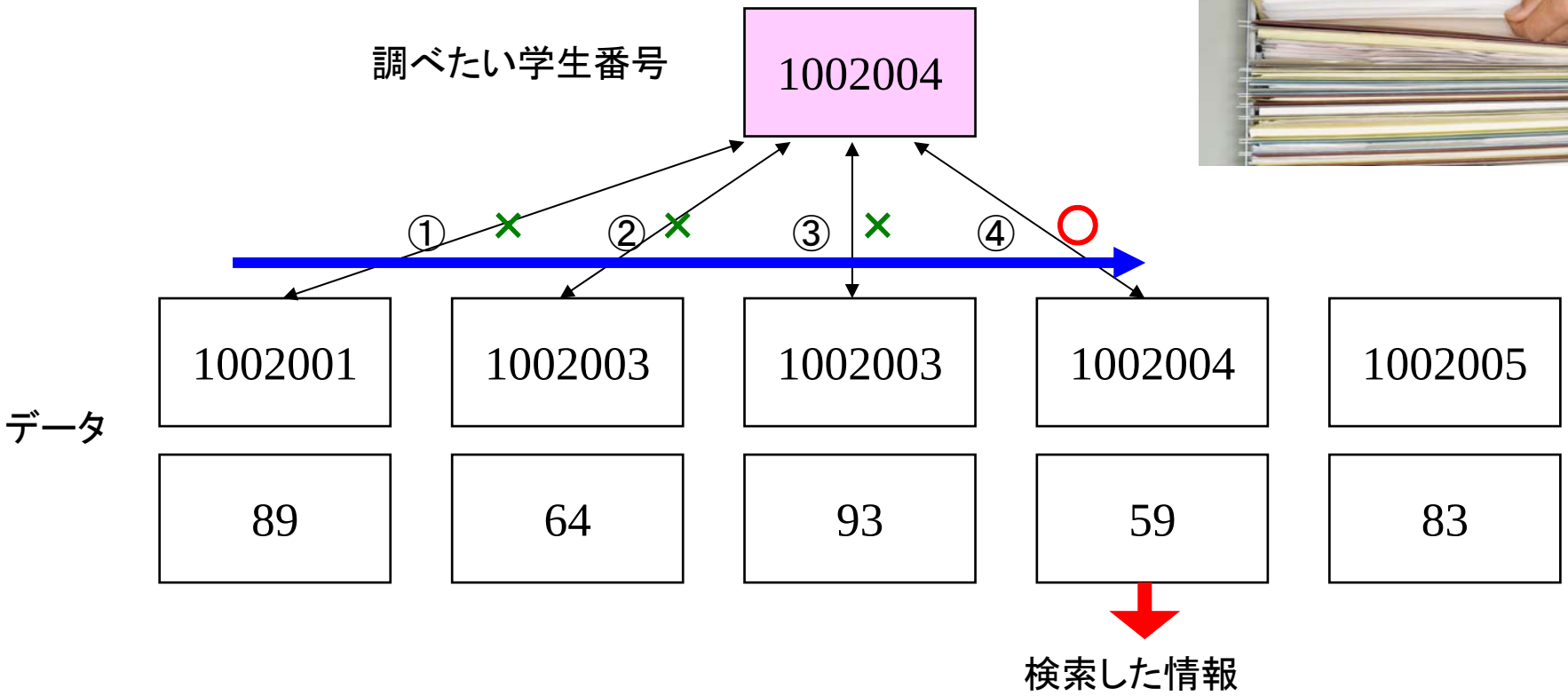
最後に
セミコロンが必要！！

- 上の例ではa[0][0]に1、a[0][1]に2、... a[2][4]に3という初期値を設定している。これは下のようにも書けるが、下の場合は要素の対応が分かり難い

```
int a[3][5] =
    {1,2,3,4,5,6,7,8,9,8,7,6,5,4,3};
```

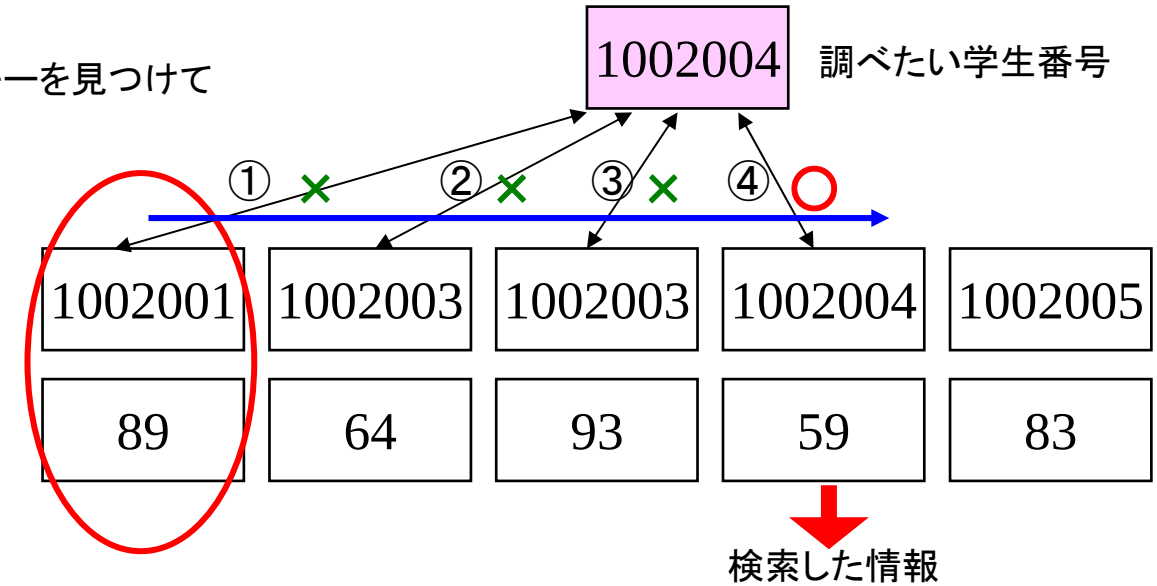
リニアサーチ

- 学生番号からその学生の得点を調べる
- いろいろな調べ方があるが、**端から順に(①、②、③の順で)調べていく方法**を「リニアサーチ」と呼ぶ。



サーチの手順

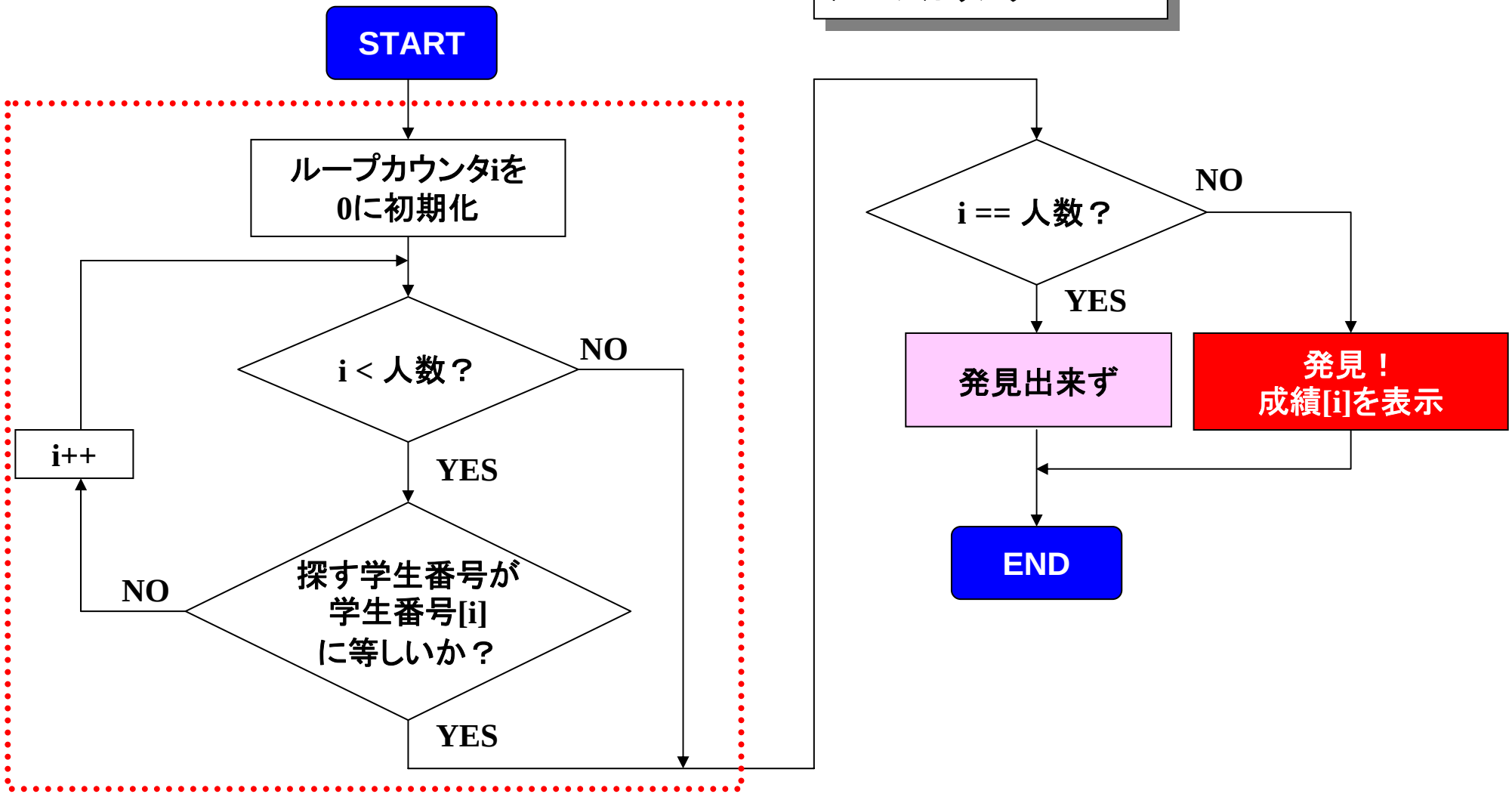
- 前ページの例で手順を説明する
- 検索されるデータは以下の赤丸のように2つ以上の情報のペアになっている。
  - 一つは検索するのに必要なデータ(この場合学生番号)で「キー」と呼ばれる
  - それ以外は検索して得たい情報である(この場合成績)
- 検索するデータはこの場合は学生番号(1002004)である
- ①②③④のように順に調べたい学生番号とキーデータとをループを使用して照合して行く。
- ①②③のように不一致である場合は次のデータとの照合を行う。もしデータがなくなったらループは終了する。
- ④のように照合が一致したら検索は終了する。
- ループが終了した時点でループの回数を調べる
  - データの個数＋1に等しい時:データの端まで来て終了したので データは見つからなかった
  - データの個数＋1より少ない時:一致するキーを見つけて 終了したので、検索したいデータを発見した
- ループは実際には0から始まるループカウンタを使用するので、上記の比較は実際にはループカウンタとデータの個数との比較となる



サーチの手順

変数

探す学生番号
人数
学生番号[人数]
成績[人数]
ループカウンタi



リニアサーチプログラム(1)

実行結果:

```
std1ss1{s1000000}1: ./a.out
Enter student ID ->1002004
   Found score : 59
Enter student ID ->234
   Not found
Enter student ID ->0
   Bye!
std1ss1{s1000000}1:
```

やってみよう

学生番号入力でおかしな入力(文字など)やEOFが入力された場合にも終了するよう作り変えてみよう

```
#include <stdio.h>
#define NUM 5

main()
{
  int  id[NUM]     = {1002001,1002002,1002003,1002004,1002005};
  int  result[NUM] = {89     ,64     ,93     ,59     ,83     };
  int  i, ID;

  while(1) /* 無限ループで検索を行う */
  {
    printf("Enter student ID ->");
    scanf("%d",&ID);
    if(ID == 0) break; /* 学生番号が0だとループ脱出 */
    i = 0;
    while (i < NUM){    /* while文での検索 */
      if(ID == id[i]) break;
      else i++;
    }
    if(i == NUM) printf("   Not found\n");      /* 発見できず */
    else printf("   Found score : %d\n",result[i]); /* 発見 */
  }
  printf("   Bye!\n"); /* 終了メッセージを表示して終了 */
}
```

リニアサーチプログラム(2)

- 検索をfor文で行う

```
#include <stdio.h>
#define NUM 5

main()
{
  int  id[NUM]     = {1002001,1002002,1002003,1002004,1002005};
  int  result[NUM] = {89     ,64     ,93     ,59     ,83     };
  int  i, ID;

  while(1) /* 無限ループで検索を行う */
  {
    printf("Enter student ID ->");
    scanf("%d",&ID);
    if(ID == 0) break; /* 学生番号が0だとループ脱出 */

    for(i = 0; (i < NUM) && (ID != id[i]); i++); /* for文での検索 */

    if(i == NUM) printf("   Not found\n");      /* 発見できず */
    else printf("   Found score : %d\n",result[i]); /* 発見 */
  }

  printf("   Bye!\n"); /* 終了メッセージを表示して終了 */
}
```

リニアサーチプログラム(3)

- 学生番号と成績を2次元配列に格納した

```
#include <stdio.h>
#define NUM 5
main()
{
  int  data[2][NUM] = {
         {1002001,1002002,1002003,1002004,1002005},
         {89     ,64     ,93     ,59     ,83     }
  }; /* 2次元配列の初期化 */
  int  i, ID;

  while(1){ /* 無限ループで検索を行う */
    printf("Enter student ID ->");
    scanf("%d",&ID);
    if(ID == 0) break; /* 学生番号が0だとループ脱出 */

    i = 0;
    while ((i < NUM) && (ID != data[0][i])) i++; /* 実際の検索 */

    if(i == NUM) printf("   Not found\n");          /* 発見できず */
    else printf("   Found score : %d\n",data[1][i]);   /* 発見 */
  }

  printf("   Bye!\n"); /* 終了メッセージを表示して終了 */
}
```

リニアサーチプログラム(4)

```
#include <stdio.h>
#define NUM 5
Int LinearSearch(int, int D[][NUM]);
main()
{
  int  data[2][NUM] = {
         {1002001,1002002,1002003,1002004,1002005},
         {89     ,64     ,93     ,59     ,83     }
  }; /* 2次元配列の初期化 */
  int  i, ID;

  while(1){ /* 無限ループで検索を行う */
    printf("Enter student ID ->");
    scanf("%d",&ID);
    if(ID = = 0) break; /* 学生番号が0だとループ脱出 */

   i=LinearSearch(ID,data);

   if(i = = NUM) printf("   Not found\n");          /* 発見できず */
    else printf("   Found score : %d\n",data[1][i]);   /* 発見 */
  }

  printf("   Bye!\n"); /* 終了メッセージを表示して終了 */
}
```

```
int LinearSearch(int ID,  int D[][NUM]){
   int i = 0;

   while ((i < NUM) && (ID != D[0][i])) i++;

   return i;
}
```