Ｃ言語の基礎Ⅱ

０．目次

１．ポインタ変数　構造体

１．１　ポインタ変数

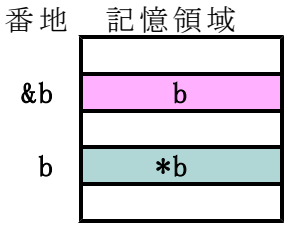
　値としてアドレスをとる変数をポインタ変数という。
int *b で bがポインタ変数であることを宣言すると、

・ポインタ変数の値を保存する記憶場所
・ポインタ変数が指す値が整数であること

が決定される。つまり、

bはポインタ変数の値、
*bはbの値が指している番地に格納されている値、
&bはポインタ変数に割り当てられた番地

を表す。

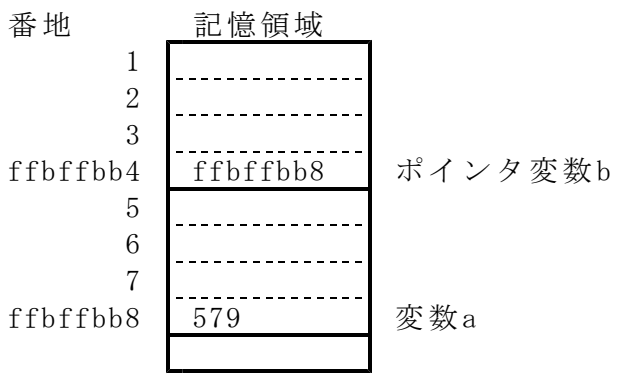


*を演算子と考える。ポインタ変数bに
演算子*を作用させ値を取得すると考える。

```
 1  /* << h111.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int a,*b; /* ポインタ変数の宣言。*/
 5    a = 123;
 6    printf(" a = %d ¥n",a);
 7    printf("&a = %x ¥n",&a);
 8    b = &a;
 9    printf(" b = %x ¥n",b);
10    printf("*b = %d ¥n",*b);
11    printf("&b = %x ¥n",&b);
12    *b = *b + 456;
13    printf("*b = %d ¥n",*b);
14    printf(" a = %d ¥n",a);
15  }
```

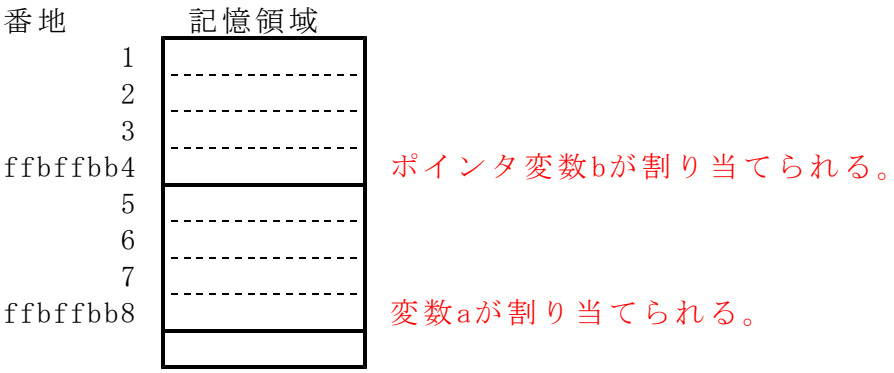
実行結果

```
1  % cc h111.c
2  % ./a.out
3   a = 123
4  &a = ffbffbb8
5   b = ffbffbb8
6  *b = 123
7  &b = ffbffbb4
8  *b = 579
9   a = 579
```

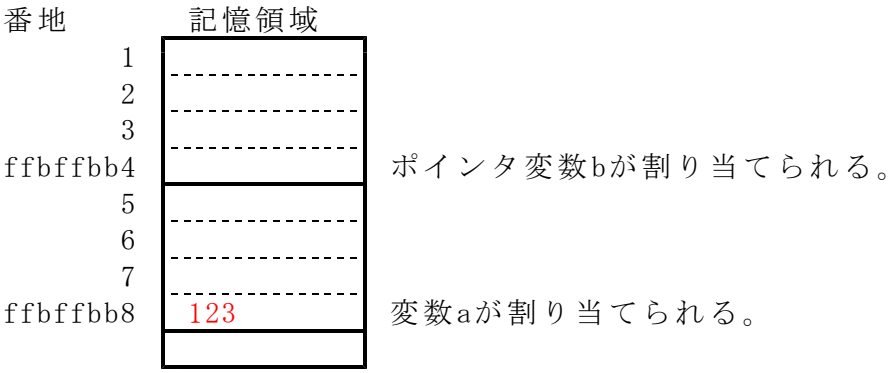


実行過程

・行番号４ ： `int a,*b;`



・行番号５ ： `a = 123;`



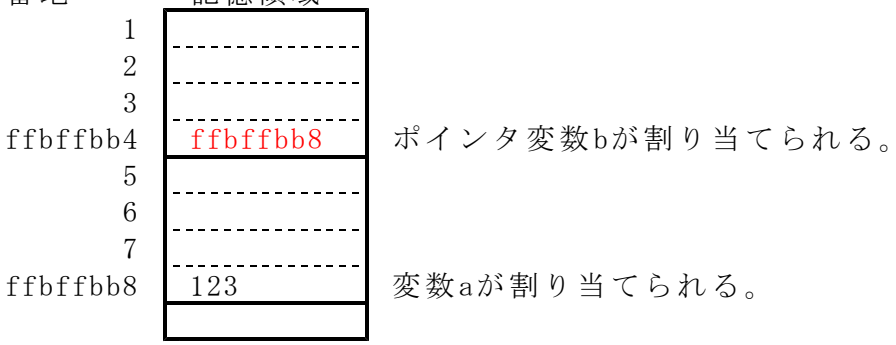
・行番号６ ： `printf(" a = %d ¥n",a);`

```
 a = 123
```

・行番号７ ： `printf("&a = %x ¥n",&a);`

```
&a = ffbffbb8
```

・行番号８ ：b = &a;



・行番号９ ：printf(" b = %x ¥n",b);

b = ffbffbb8

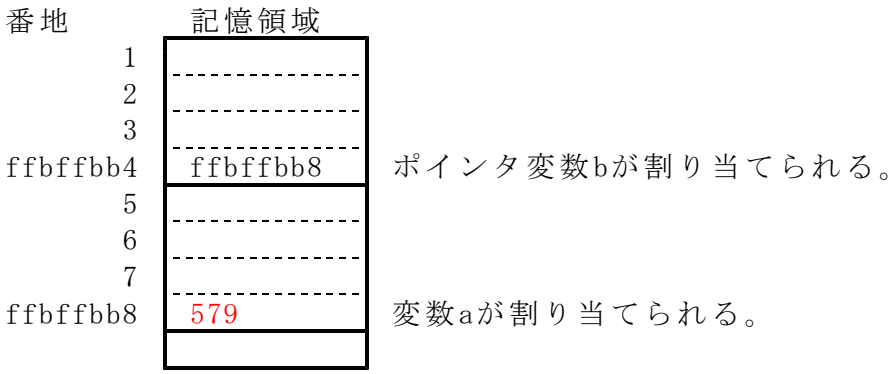
・行番号１０ ：printf("*b = %d ¥n",*b);

*b = 123

・行番号１１ ：printf("&b = %x ¥n",&b);

&b = ffbffbb4

・行番号１２ ：*b = *b + 456;



・行番号１３ ：printf("*b = %d ¥n",*b);

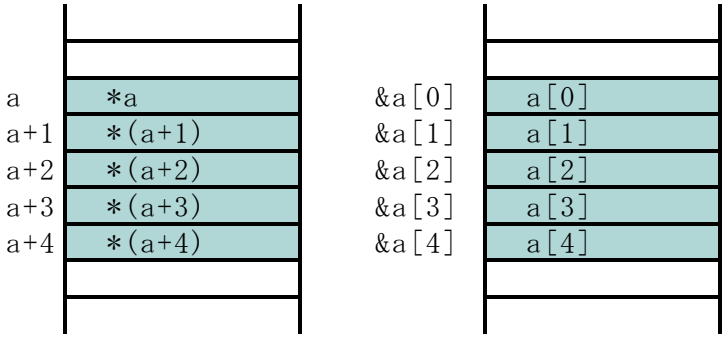
*b = 579

・行番号１４ ：printf(" a = %d ¥n",a);

a = 579

１．２　ポインタ変数と配列

int a[5] と宣言すると、配列名aは配列の先頭アドレスを指すポインタを表す。したがって、a[0]と*a,a[1]と*(a+1),a[2]と*(a+2)は同じ値を持つ。




```
 1  /* << h121.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int a[3],i;
 5    for( i=0; i<3; i++ ) { a[i] = 100 * i; }
 6    for( i=0; i<3; i++ ) {
 7      printf("a[%d] = %3d   &a[%d] = %x ¥n",i,a[i],i,&a[i]);
 8    }
 9    for( i=0; i<3; i++ ) {
10      printf("*(a+%d) = %3d   a+%d = %x ¥n",i,*(a+i),i,a+i);
11    }
12  }
```

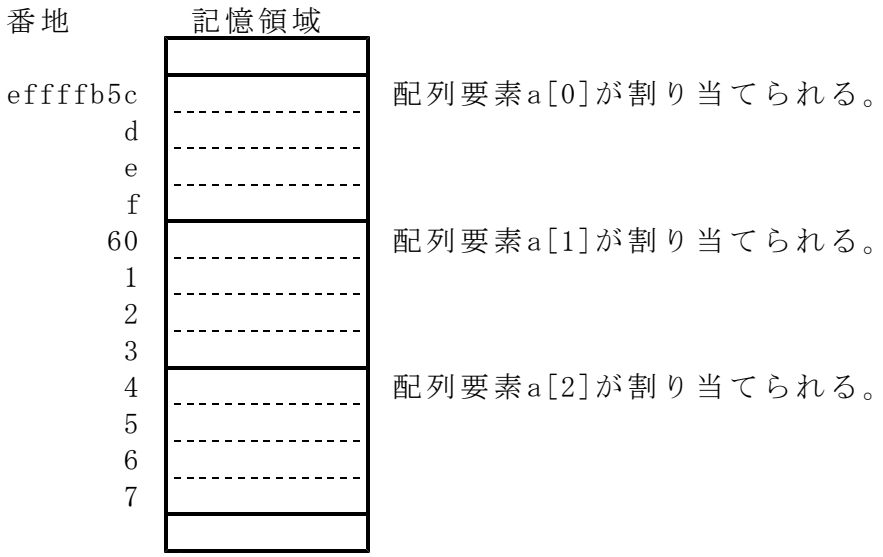
実行結果

```
1  % cc h121.c
2  % ./a.out
3  a[0] =   0   &a[0] = effffb5c
4  a[1] = 100   &a[1] = effffb60
5  a[2] = 200   &a[2] = effffb64
6  *(a+0) =   0   a+0 = effffb5c
7  *(a+1) = 100   a+1 = effffb60
8  *(a+2) = 200   a+2 = effffb64
```

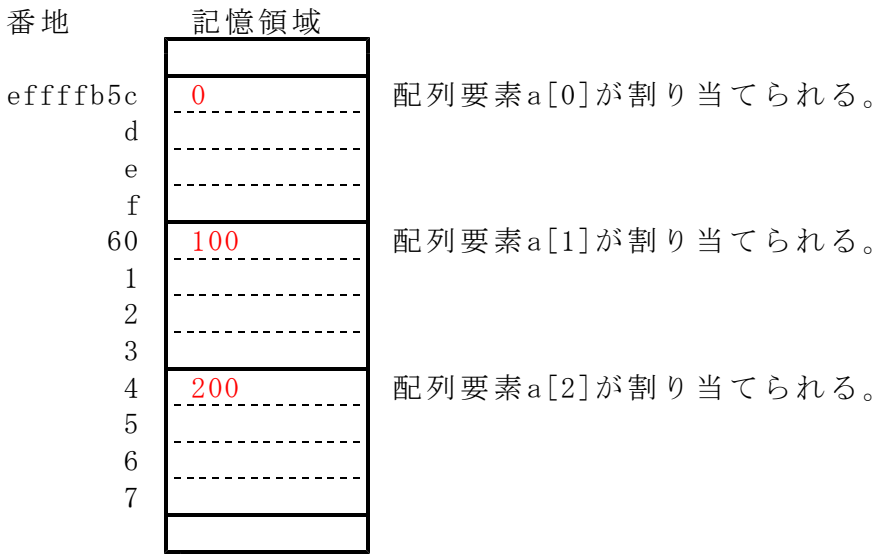
（注意）4バイト分増加している。

実行過程

・行番号４： `int a[3],i;`



・行番号５： `for( i=0; i<3; i++ ) { a[i] = 100 * i; }`



・行番号６，７，８：
```
for( i=0; i<3; i++ ) {
    printf("a[%d] = %3d   &a[%d] = %x ¥n",i,a[i],i,&a[i]);
}
```

```
a[0] =   0   &a[0] = effffb5c
a[1] = 100   &a[1] = effffb60
a[2] = 200   &a[2] = effffb64
```

・行番号９，１０，１１：
```
for( i=0; i<3; i++ ) {
    printf("*(a+%d) = %3d    a+%d = %x ¥n",i,*(a+i),i,a+i);
}
```

```
*(a+0) =   0    a+0 = effffb5c
*(a+1) = 100    a+1 = effffb60
*(a+2) = 200    a+2 = effffb64
```

配列名aに代入はできない。

```
1  /* << h122.c >> */
2  #include <stdio.h>
3  int main() {
4    int a[3],*b;
5    b = a;
6    a = b;
7  }
```

実行結果

```
% cc h122.c
h122.c: In function ‘main’:
h122.c:6: error: incompatible types when assigning to type ‘in
t[3]’ from type ‘int *’
```

１．３　構造体

ひとまとまりのデータを扱いやすくするために、いろいろな型（int型,float型,char型など）のデータを集めたものを構造体という。構造体はいくつかのメンバから構成され、各メンバはメンバ名と型をもつ。

```
struct {
  型1  メンバ１;
  型2  メンバ２;
}
構造体名;
```

同じ構造体を複数個使うときは、タグ（名前）を使って構造体型を宣言しておき、それを利用できる。

```
struct タグ {
  型1  メンバ１;
  型2  メンバ２;
};
struct タグ 構造体名;
```

●複素数を構造体で表現してみる。

```
 1  /* << h131.c >> */
 2  #include <stdio.h>
 3  /* 構造体型の宣言。タグをcomplexとする。*/
 4  struct complex {
 5    float re; /* メンバre */
 6    float im; /* メンバim */
 7  };
 8  int main() {
 9    struct complex x,y,z; /* 構造体の宣言。*/
10    /* メンバの参照は、構造体名とメンバ名を.でつなぐ。*/
11    x.re = 11.1; x.im = 22.2;
12    y.re = -33.3; y.im = -44.4;
13    z.re = x.re + y.re;
14    z.im = x.im + y.im;
15    printf("x.re = %8.3f    x.im = %8.3f ¥n",x.re,x.im);
16    printf("y.re = %8.3f    y.im = %8.3f ¥n",y.re,y.im);
17    printf("z.re = %8.3f    z.im = %8.3f ¥n",z.re,z.im);
18  }
```

実行結果

```
% cc h131.c
% ./a.out
x.re =   11.100    x.im =   22.200
y.re =  -33.300    y.im =  -44.400
z.re =  -22.200    z.im =  -22.200
```

●構造体の配列を表現できる。

```
/* << h132.c >> */
#include <stdio.h>
/* 構造体型の宣言。タグをcomplexとする。*/
struct complex {
  float re; /* メンバre */
  float im; /* メンバim */
};
int main() {
  int i;
  struct complex z[9]; /* 構造体配列の宣言。*/
  z[0].re = 11.1; z[0].im = 22.2;
  z[1].re = -33.3; z[1].im = -44.4;
  z[2].re = z[0].re + z[1].re;
  z[2].im = z[0].im + z[1].im;
  for( i=0; i<=2; i++ ) {
    printf("z[%d].re = %8.3f   z[%d].im = %8.3f ¥n",
            i,z[i].re,i,z[i].im);
  }
}
```

実行結果

```
% cc h132.c
% ./a.out
z[0].re =   11.100    z[0].im =   22.200
z[1].re =  -33.300    z[1].im =  -44.400
z[2].re =  -22.200    z[2].im =  -22.200
```

２．関数

ひとつの問題を解く場合、その問題を機能別に分解して解くと、解法がわかりやすく、改良しやすくなる。このようなことを可能にする方法として関数がある。

関数を使う場合、**関数宣言、関数定義、関数呼出し**を行う。

●関数宣言

関数宣言では、戻り値(関数が返す値)の型、関数の名前、仮引数(型 変数名)と書く。戻り値がないときは、void と書く。

```
戻り値の型　関数名(仮引数，…，仮引数)
```

●関数定義

関数定義は一般的につぎのように書く。

```
戻り値の型　関数名（仮引数，…，仮引数）{
　文
}
```

●関数呼出し

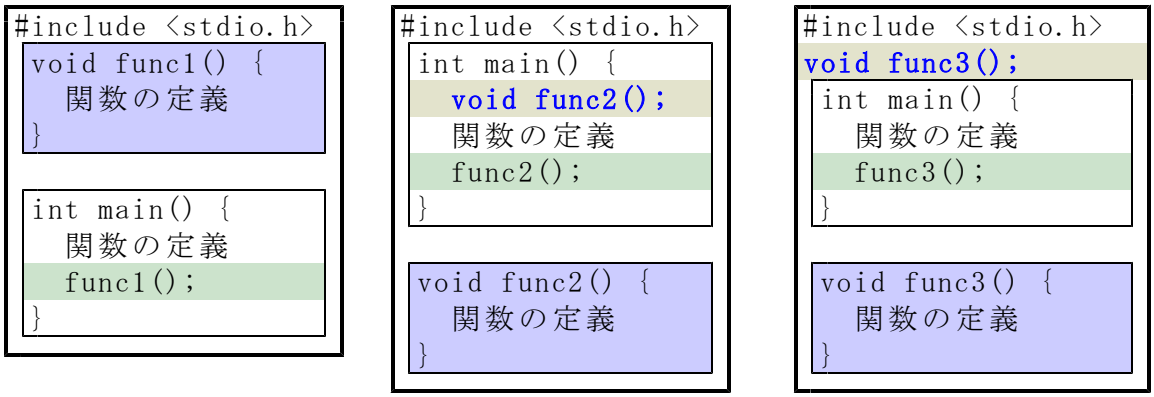
関数呼出しでは、関数名と実引数を書く。

```
関数名（実引数，…，実引数）
```

プログラム中で関数が呼出されると、実引数が評価され、**その値が対応する仮引数に与えられる。**そして、この関数が実行され終了すると、関数が呼出されたところへ戻る。

（注意１）実引数と仮引数はその個数と型は一致していなければならない。

（注意２）main関数以外の関数を使う場合、main関数が書かれる前にその関数を定義する必要がある。main関数以降に定義する場合、あらかじめ関数宣言をする必要がある。



２．１　戻り値がない場合

voidは戻り値がないことを示す。

```
 1  /* << h211.c >> */
 2  #include <stdio.h>
 3  int main() {
 4    int a,b;
 5    void func(int x, int y); /* 関数の宣言 */
 6    a = 11;
 7    b = 22;
 8    func(a,b); /* 関数の呼出し */
 9    func(a+b,a*b); /* 関数の呼出し */
10  }
11  void func(int x, int y) { /* 関数の定義 */
12    printf("%d + %d = %d ¥n",x,y,x+y);
13  }
```

実行結果

```
1  % cc h211.c
2  % ./a.out
3  11 + 22 = 33
4  33 + 242 = 275
```

２．２　戻り値がある場合

```
1  /* << h221.c >> */
2  #include <stdio.h>
3  int main() {
4    int a,b,c;
5    int func(int x, int y); /* 関数の宣言 */
6    a = 11;
7    b = 22;
8    c = func(a,b); /* 関数の呼出し */
9    printf("%d + %d = %d ¥n",a,b,c);
10   c = func(a+b,a*b); /* 関数の呼出し */
11   printf("%d + %d = %d ¥n",a+b,a*b,c);
12 }
13 int func(int x, int y) { /* 関数の定義 */
14   int z;
15   z = x + y;
16   return z; /* return文で、zの値が戻り値の型に変換され、
17                戻り値として戻される。*/
18 }
```

実行結果

```
1  % cc h221.c
2  % ./a.out
3  11 + 22 = 33
4  33 + 242 = 275
```

２．３　引数について

●値呼出し

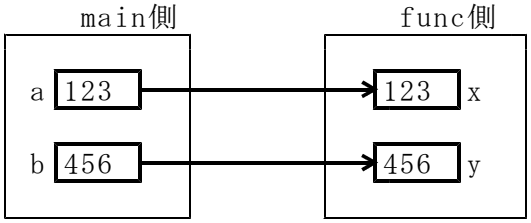
変数の場合、値が関数側に渡される。したがって、関数内で対応する変数が更新されても呼出し側は影響を受けない。**値呼出し**という。
仮引数や関数の内部で宣言された変数は**局所変数**と呼ばれる。

```
 1  /* << h231.c >> */
 2  #include <stdio.h>
 3  int main () {
 4    int a,b;
 5    void func(int x, int y);
 6    a = 123; b = 456;
 7    printf("main1: a=%d b=%d¥n",a,b);
 8    func(a,b); /* 関数の呼出し。*/
 9    printf("main2: a=%d b=%d¥n",a,b);
10  }
11  void func(int x, int y) {
12    printf("func1: x=%d y=%d¥n",x,y);
13    x = -x; y = -y; /* 変数x,yの更新。*/
14    printf("func2: x=%d y=%d¥n",x,y);
15  }
```

実行結果

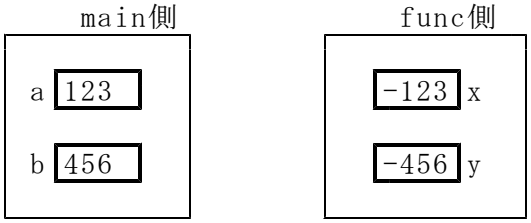
```
% cc h231.c
% ./a.out
main1: a=123 b=456
func1: x=123 y=456
func2: x=-123 y=-456
main2: a=123 b=456
```

（１）関数開始時

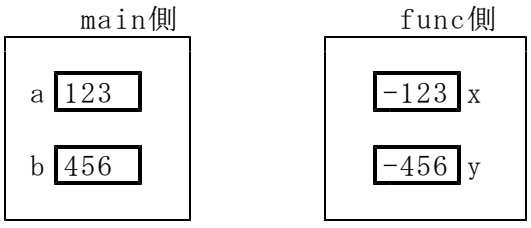


変数x,yに記憶場所が確保され、呼出す側（main）の変数a,bの値が関数側（func）の変数x,yへ代入される。

（２）関数内処理



（３）関数終了時



関数側（func）の変数x,yは解放される。

●変数呼出し

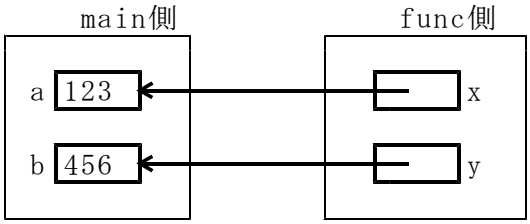
ポインタ変数を使うと、関数内での変数値変更を呼出し側に及ぼすことができる。**変数呼出し**という。同じ例で示す。

```
 1  /* << h232.c >> */
 2  #include <stdio.h>
 3  int main () {
 4    int a,b;
 5    void func(int *x, int *y);
 6    a = 123; b = 456;
 7    printf("main1: a=%d b=%d¥n",a,b);
 8    func(&a,&b); /* アドレスを実引数と
 9                    する関数の呼出し。*/
10    printf("main2: a=%d b=%d¥n",a,b);
11  }
12  void func(int *x, int *y) {
13    printf("func1: *x=%d *y=%d¥n",*x,*y);
14    *x = -*x; *y = -*y;/* ポインタ変数
15                          x,yの更新 */
16    printf("func2: *x=%d *y=%d¥n",*x,*y);
17  }
```

実行結果

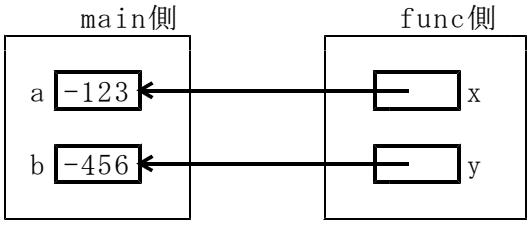
```
% cc h232.c
% ./a.out
main1: a=123 b=456
func1: *x=123 *y=456
func2: *x=-123 *y=-456
main2: a=-123 b=-456
```

（１）関数開始時

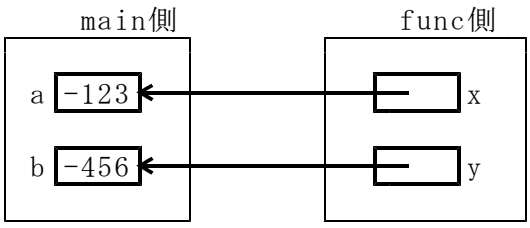


ポインタ変数x,yに記憶場所が確保され、呼出す側（main）の変数a,bのアドレスが関数側（func）のポインタ変数x,yへ代入される。

（２）関数内処理



（３）関数終了時



関数側（func）のポインタ変数x,yは解放される。

●例１

```
 1  /* << h233a.c >> */
 2  #include <stdio.h>
 3  void func(int a, int b);
 4  int main () {
 5    int a,b;
 6    while( 1 ) {
 7      scanf("%d%d",&a,&b);
 8      if( (a <= 0)||(b <= 0) ) { break; }
 9      printf("%d %d¥n",a,b);
10      func(a,b);
11      printf("%d %d¥n",a,b);
12    }
13  }
14  void func(int a, int b) {
15    int w;
16    w = a; a = b; b = w;
17    printf("%d %d ",a,b);
18  }
```

実行結果

```
% cc h233a.c
% ./a.out
111 222
111 222
222 111 111 222
```

●例２

```
 1  /* <<h233b.c>> */
 2  #include <stdio.h>
 3  void func(int *a, int *b);
 4  int main () {
 5    int a,b;
 6    while( 1 ) {
 7      scanf("%d%d",&a,&b);
 8      if( (a <= 0)||(b <= 0) ) { break; }
 9      printf("%d %d¥n",a,b);
10      func(&a,&b);
11      printf("%d %d¥n",a,b);
12    }
13  }
14  void func(int *a, int *b) {
15    int w;
16    w = *a; *a = *b; *b = w;
17    printf("%d %d ",*a,*b);
18  }
```

実行結果

```
% cc h233b.c
% ./a.out
111 222
111 222
222 111 222 111
```

●例３

```
 1  /* << h233c.c >> */
 2  #include <stdio.h>
 3  void func(int *a, int *b);
 4  int main () {
 5    int a,b;
 6    while( 1 ) {
 7      scanf("%d%d",&a,&b);
 8      if( (a <= 0)||(b <= 0) ) { break; }
 9      printf("%d %d¥n",a,b);
10      func(&a,&b);
11      printf("%d %d¥n",a,b);
12    }
13  }
14  void func(int *a, int *b) {
15    int *w;
16    w = a; a = b; b = w;
17    printf("%d %d ",*a,*b);
18  }
```

実行結果

```
% cc h233c.c
% ./a.out
111 222
111 222
222 111 111 222
```

●局所変数は、関数が呼ばれると記憶場所を割り当てられ、関数が終了すると解放される。したがって、処理結果をつぎの関数呼出しまで保存できない。保存するためには、関数の外で変数を宣言する。この変数を**大域変数**といい、どの関数からでも代入、参照ができる。

```
1  /* << h234.c >> */
2  #include <stdio.h>
3  int x; /* 大域変数。*/
4  void func1(void);
5  void func2(void);
6  int main () {
7    func1();
8    func2();
9  }
10 void func1(void) {
11   x = 123;
12   printf("(func1):x = %d¥n",x);
13 }
14 void func2(void) {
15   x = x * 2;
16   printf("(func2):x = %d¥n",x);
17 }
```

実行結果

```
1  % cc h234.c
2  % ./a.out
3  (func1):x = 123
4  (func2):x = 246
```

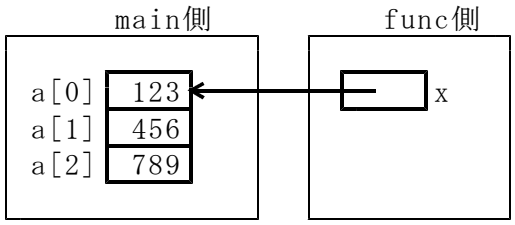
●配列の場合、アドレスが関数側に渡される。したがって、関数内で対応する配列が更新されると呼出し側は影響を受ける。

```
1  /* << h235.c >> *
2  #include <stdio.h>
3  void func(int x[]);
4  int main () {
5    int a[3];
6    a[0] = 123; a[1] = 456; a[2] = 789;
7    printf("main1: a[0]=%d a[1]=%d a[2]=%d¥n",a[0],a[1],a[2]);
8    func(a); /* 配列を実引数とする関数の呼出し。*/
9    printf("main2: a[0]=%d a[1]=%d a[2]=%d¥n",a[0],a[1],a[2]);
10 }
11 void func(int x[]) {
12   printf("func1: x[0]=%d x[1]=%d x[2]=%d¥n",x[0],x[1],x[2]);
13   x[0] = -x[0]; x[1] = -x[1]; x[2] = -x[2]; /* 配列xの更新。*/
14   printf("func2: x[0]=%d x[1]=%d x[2]=%d¥n",x[0],x[1],x[2]);
15 }
```

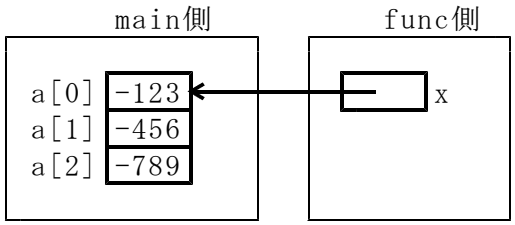
実行結果

```
1  % cc h235.c
2  % a.out
3  main1: a[0]=123 a[1]=456 a[2]=789
4  func1: x[0]=123 x[1]=456 x[2]=789
5  func2: x[0]=-123 x[1]=-456 x[2]=-789
6  main2: a[0]=-123 a[1]=-456 a[2]=-789
```

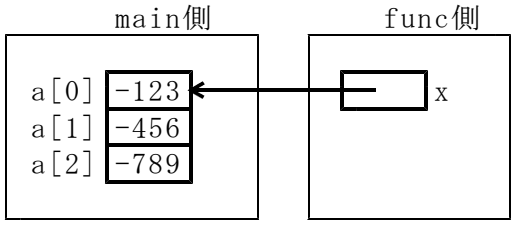
（１）関数開始時



（２）関数内処理



（３）関数終了時



（注意）配列aの記憶場所は確保されるが配列xの記憶場所は確保されない。
関数呼出し時に、呼出す側（main）の配列のアドレスが
関数側（func）に渡され記憶場所は共有される。

●２次元配列の場合もアドレスが関数側に渡される。したがって、関数内で対応する配列が更新されると呼出し側は影響を受ける。

```
/* << h236.c >> */
#include <stdio.h>
void func(int x[][3]);
int main () {
  int a[2][3];
  a[0][0] = 11; a[0][1] = 12; a[0][2] = 13;
  a[1][0] = 21; a[1][1] = 22; a[1][2] = 23;
  printf("main1:¥n");
  printf("a[0][0]=%d a[0][1]=%d a[0][2]=%d¥n",a[0][0],a[0][1],a[0][2]);
  printf("a[1][0]=%d a[1][1]=%d a[1][2]=%d¥n",a[1][0],a[1][1],a[1][2]);
  func(a); /* 配列を実引数とする関数の呼出し。*/
  printf("main2:¥n");
  printf("a[0][0]=%d a[0][1]=%d a[0][2]=%d¥n",a[0][0],a[0][1],a[0][2]);
  printf("a[1][0]=%d a[1][1]=%d a[1][2]=%d¥n",a[1][0],a[1][1],a[1][2]);
}
void func(int x[][3]) {
  printf("func1:¥n");
  printf("x[0][0]=%d x[0][1]=%d x[0][2]=%d¥n",x[0][0],x[0][1],x[0][2]);
  printf("x[1][0]=%d x[1][1]=%d x[1][2]=%d¥n",x[1][0],x[1][1],x[1][2]);
  /* 配列xの更新。*/
  x[0][0] = -x[0][0]; x[0][1] = -x[0][1]; x[0][2] = -x[0][2];
  x[1][0] = -x[1][0]; x[1][1] = -x[1][1]; x[1][2] = -x[1][2];
  printf("func2:¥n");
  printf("x[0][0]=%d x[0][1]=%d x[0][2]=%d¥n",x[0][0],x[0][1],x[0][2]);
  printf("x[1][0]=%d x[1][1]=%d x[1][2]=%d¥n",x[1][0],x[1][1],x[1][2]);
}
```

実行結果

```
% cc h236.c
% ./a.out
main1:
a[0][0]=11 a[0][1]=12 a[0][2]=13
a[1][0]=21 a[1][1]=22 a[1][2]=23
func1:
x[0][0]=11 x[0][1]=12 x[0][2]=13
x[1][0]=21 x[1][1]=22 x[1][2]=23
func2:
x[0][0]=-11 x[0][1]=-12 x[0][2]=-13
x[1][0]=-21 x[1][1]=-22 x[1][2]=-23
main2:
a[0][0]=-11 a[0][1]=-12 a[0][2]=-13
a[1][0]=-21 a[1][1]=-22 a[1][2]=-23
```