第１１回ポインタ

1

今回の目標

- C言語におけるポインタを理解する。
- アドレス演算子、参照演算子の効果を理解する。
- 参照による関数呼び出しを理解する。
- 配列とポインタの関係を理解する。

☆他の関数内の２つの変数の値を交換する関数を作成し、その効果を確かめるプログラムを作成する。

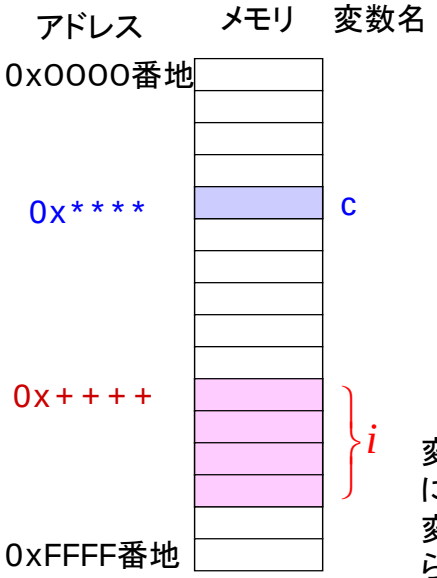
2

ポインタ

ポインタとは、
変数のアドレスを入れる変数である。

3

アドレス



コンピュータのメモリには、
すべてアドレスがある。

C言語を用いたプログラムでは、
プログラマがアドレスを管理できる。

```
char  c;
```

1バイト

```
int  i;
```

4バイト

変数宣言すると、その変数のために
メモリが割り当てられる。
変数名は、メモリの一区画につけられた名前である。

4

アドレス演算子　&

C言語には、変数に割り当てられたメモリの、
先頭アドレスを陽に調べる演算子がある。

書式

```
& 変数名
```

&は、変数のアドレスを求めるための単項演算子。

例

```
int   age;
scanf("%d",&age);
```

```
double   height;
scanf("%lf",&height);
```

実は、scanf(の変換仕様)では、変数のアドレス
を指定すると、そのアドレスが割り当てられてい
る変数（メモリ）に標準入力から値を読み込む。

5

アドレスとscanf文

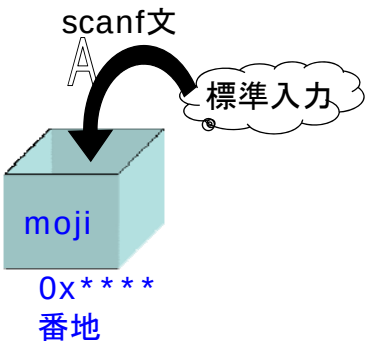
書式

```
scanf("%c",文字をいれる変数のアドレス）
scanf("%d",整数をいれる変数のアドレス）
scanf("%lf",実数をいれる変数のアドレス）
```

例

```
char   moji;
scanf("%c",&moji);
```

&がついているので
アドレス。



6

アドレスとprintf文

printf文には、アドレスを表示するための変換仕様がある。

%p ⟷ アドレス
(型の区別無し)

例

変数のアドレスを表示させるには、

```
char   moji;
printf("%p",&moji);
```

16進数で表示される。

&がついているので
アドレス

変数の中身(値)を表示させるには、

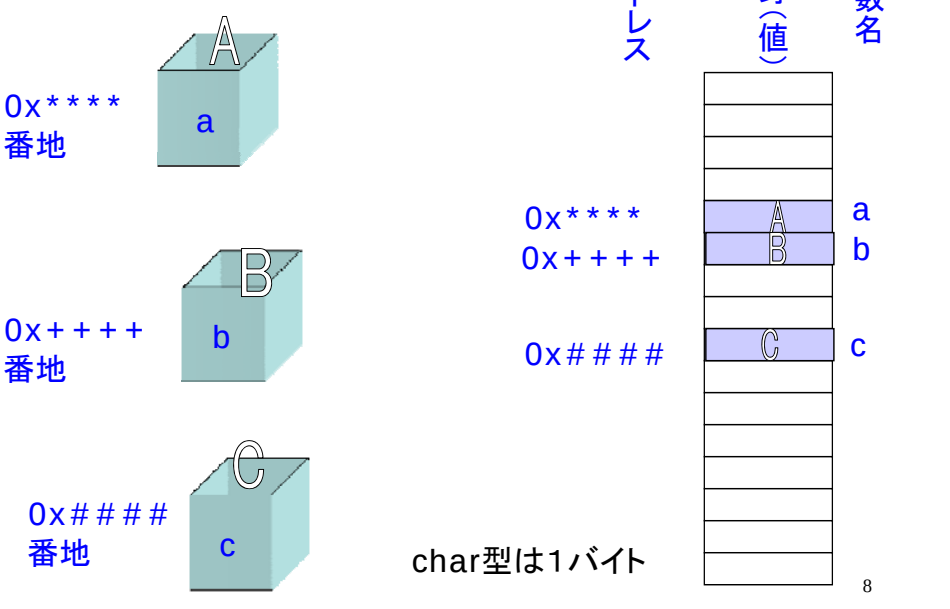
```
char   moji;
printf("%c",moji);
```

文字として表示される。

&がついていないので
中身(値)

7

イメージ

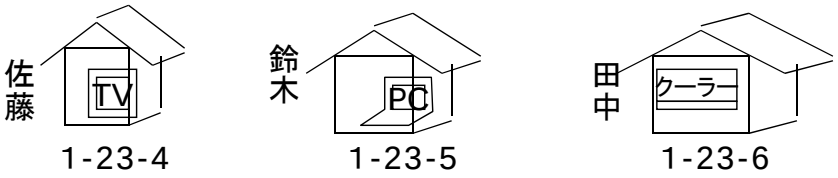


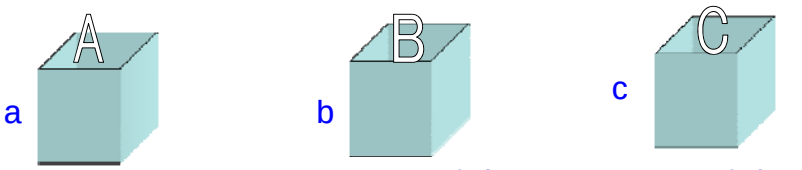
8

イメージ

入れ物(建物)における
名前と
番地(住所)と
中に入っている物

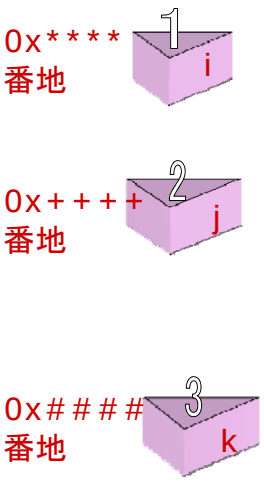
変数における
変数名と
アドレスと
中に入っている値

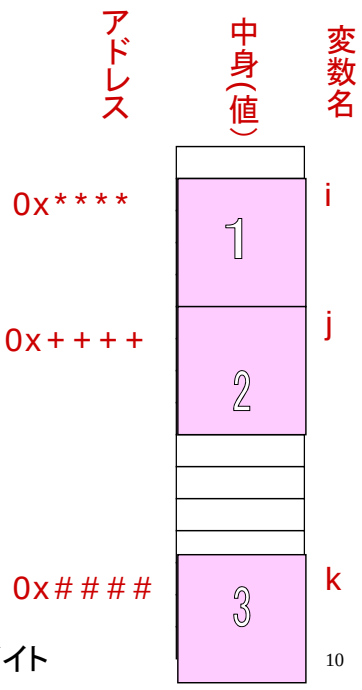




9

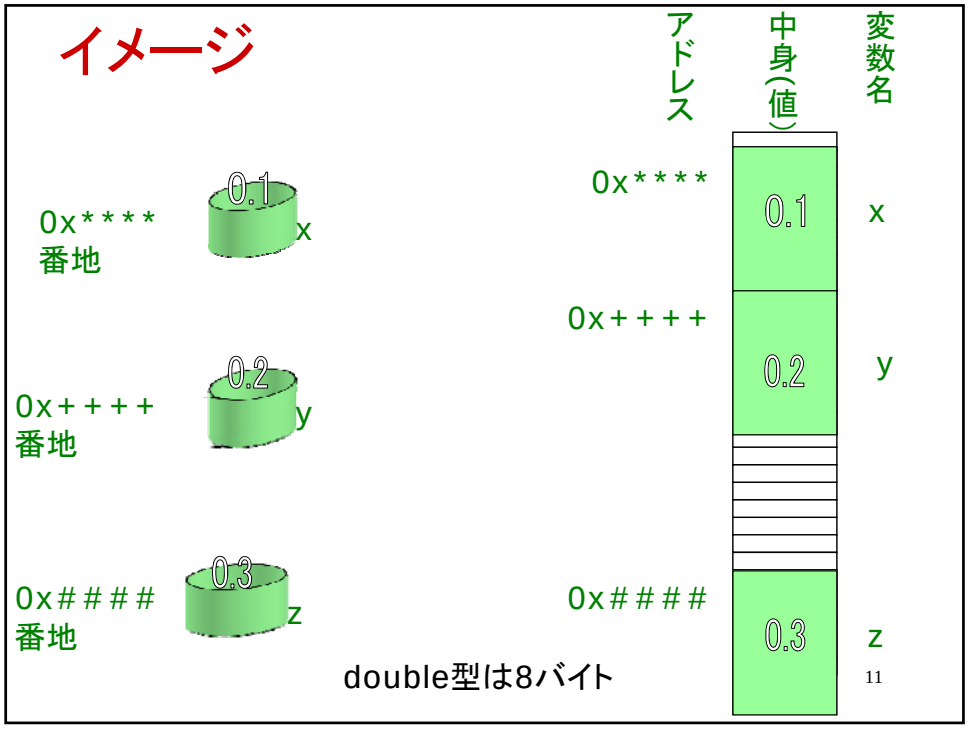
イメージ





int型は4バイト

10



練習1

```
/* print_address.c  アドレス表示実験      */
#include <stdio.h>
int main()
{    /*変数宣言*/
        char  a;
        char  b;
        int     i;
        int     j;
        double x;
        double y;
        /*代入*/
        a='A';
        b='B';
        i=1;
        j=2;
        x=0.1;
        y=0.2;
        /*      次へ続く        */
```

```
        printf("char 型の変数のアドレス¥n");
        printf("a:%p  b:%p  ¥n",&a,&b);
        printf("char型の変数の中身¥n");
        printf("a:%c   b:%c   ¥n",a,b);
        printf("¥n");

        printf("int型の変数のアドレス¥n");
        printf("i:%p   j:%p ¥n",&i,&j);
        printf("int型の変数の中身¥n");
        printf("i:%d   j:%d ¥n",i,j);
        printf("¥n");

        printf("double型の変数のアドレス¥n");
        printf("x:%p  x:%p ¥n",&x,&y);
        printf("double型の変数の中身¥n");
        printf("x:%f   y:%f ¥n",x,y);
        return 0;
}
```

ポインタの宣言

変数のアドレスを入れるための変数（ポインタ）の用意の仕方。

宣言

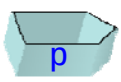
```
データ型　　＊ポインタの名前；
```

例

```
char  *p;
```

```
int  *q;
```

```
double  *r;
```



文字型の変数の
アドレス専用

整数型の変数の
アドレス専用

実数型の変数の
アドレス専用

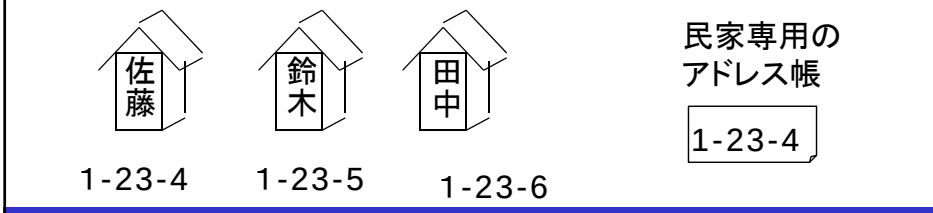
ポインタ＝変数のアドレスを入れるための入れ物
（ただし、用途別）

イメージ

（種類別の）建物
住所
（種類別の）アドレス帳

対応

変数（の型）
アドレス
（ある型の変数を指す）ポインタ



15

間接演算子　*
（ポインタとポインタが指す変数）

ポインタに、変数のアドレスを代入すると、
間接演算子＊でそのポインタが指す
変数の値を参照できる。

例

```
int     i;
int     *p;  /*ポインタ*/

p=(&i);
/*  pにはi のアドレスが入る*/
```

ポインタpがある変数yのアドレスを蓄えているとき、
ポインタpは変数yを指すという。
あるいは、pは変数yへのポインタであるという。

16

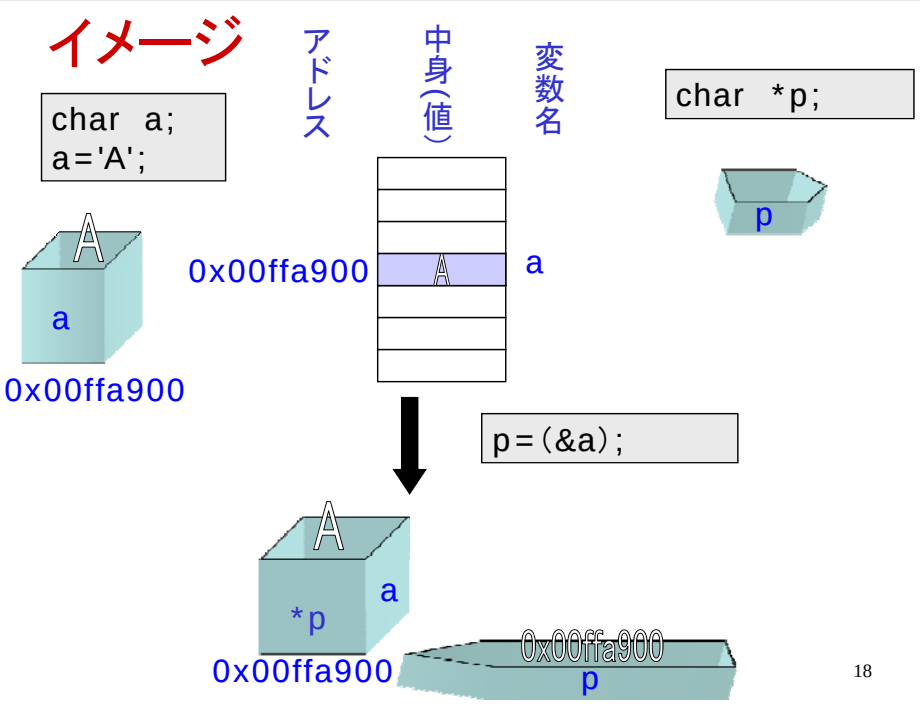
ポインタと変数の別名

ポインタpがある変数yのアドレスを蓄えているとき、
(*p)はあたかも変数yのように振舞う。

```
char  a;
char  b;
char  *p;

p=(&a);/* pはaを指す。*/

b=(*p);
/*これは「b=a;」と同じ。*/

(*p)=b;
/*これは「a=b;」と同じ。*/
```

17

イメージ



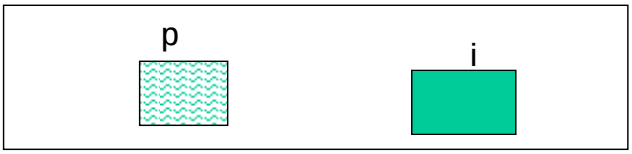
18

イメージ

プログラムを図で説明するときには、
アドレスを数字で表さずに、
矢印でポインタをあらわすことがある。

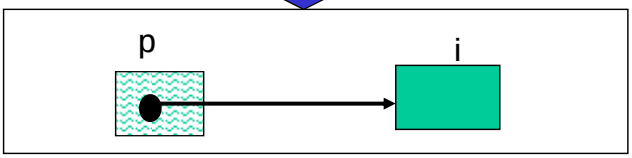
変数宣言（変数の用意）

```
int i;
int *p;
```



アドレス代入

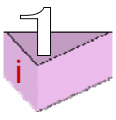
```
p=(&i);
```



19

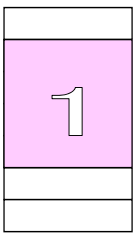
イメージ

```
int  i;
i=1;
```

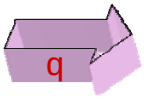


0x00ffbba8

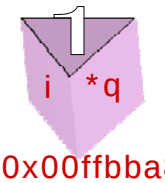
アドレス　中身（値）　変数名



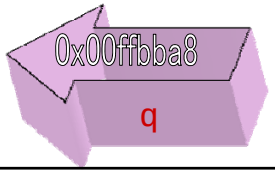

```
int  *q;
```



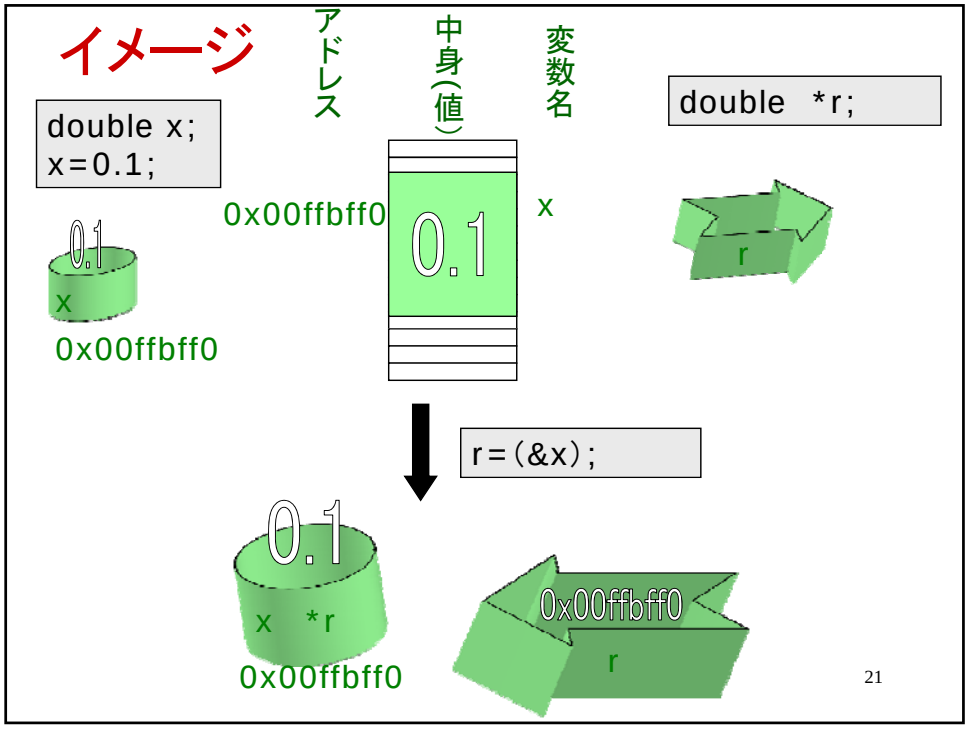

```
q=(&i);
```



0x00ffbba8



20



NULL

C言語では、どの変数も指さない特別なアドレスがあり、
NULL
として表す。(スタイル規則参照)

```
int    *p;    /*int型を指すポインタ*/
double *q; /*double型を指すポインタ*/

/*ポインタへNULLを代入*/
p=NULL;
q=NULL;
```

このように、
どんな型の変数を指す
ポインタへもNULLを代入
できます。

22

練習２

```
/* test_pointer.c   ポインター実験  コメント省略    */
#include <stdio.h>
int main()
{
        /*変数宣言*/
        int     i;    /*整数が入る変数*/
        int     j;
        int     *p;  /*アドレスが入る変数（ポインタ）*/
        int     *q;

        /*代入*/
        i=1;
        j=2;

        /*      次へ続く        */
```

23

```
        /*続き*/
        /*実験操作*/
        p=(&i);         /*ポインタpへ変数iのアドレスを代入*/
        q=(&j);         /*ポインタqへ変数jのアドレスを代入*/

        /*続き*/
        printf("アドレス代入直後¥n");

        printf("iの中身は、%d¥n",i);
        printf("iのアドレスは、%p¥n",&i);
        printf("pの中身は、%p¥n",p);
        printf("pの指す変数の中身は、%d¥n¥n",*p);

        printf("jの中身は、%d¥n",j);
        printf("jのアドレスは、%p¥n",&j);
        printf("qの中身は、%p¥n",q);
        printf("qの指す変数の中身は、%d¥n¥n¥n",*q);
        /*      次へ続く*/
```

24

```
        /*続き*/
        /*ポインタによる演算*/
        (*q)=(*q)+(*p);
        printf("(*q)=(*q)+(*p);実行¥n");
        printf("¥n");

        printf("iの中身は、%d¥n",i);
        printf("iのアドレスは、%p¥n",&i);
        printf("pの中身は、%p¥n",p);
        printf("pの指す変数の中身は、%d¥n",*p);
        printf("¥n¥n");

        printf("jの中身は、%d¥n",j);
        printf("jのアドレスは、%p¥n",&j);
        printf("qの中身は、%p¥n",q);
        printf("qの指す変数の中身は、%d¥n",*q);

        return 0;
}
```

25

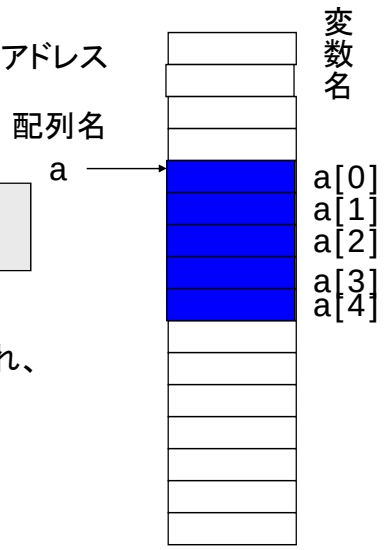
配列とポインタ

C言語では、配列名は先頭の要素のアドレスを指す。

例えば、

```
#define  MAX  5
char  a[MAX];
```

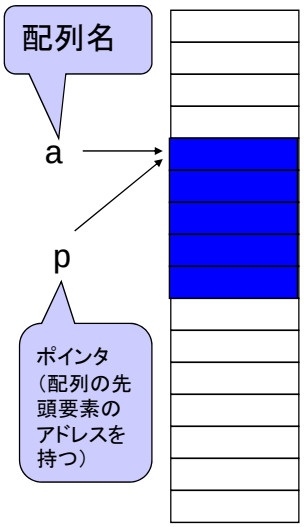
と宣言するとアドレスの連続した
5個のchar変数がメモリ上に確保され、
その先頭のアドレスがaにはいる。
つまり、aには「&a[0]の値（アドレス）」が保持されている。



26

```
char a[MAX];
char *p;
p=a;
```

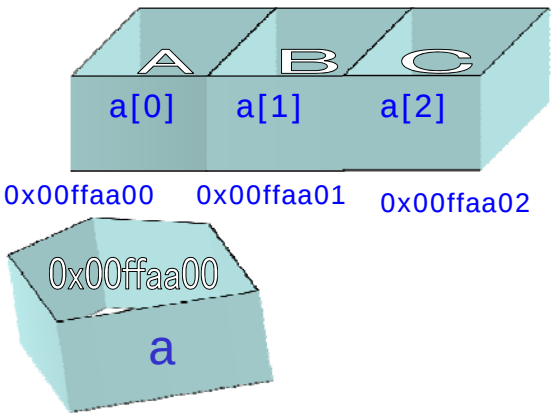
とすると、
a[i]とp[i]は同じ変数（配列要素）を表す。

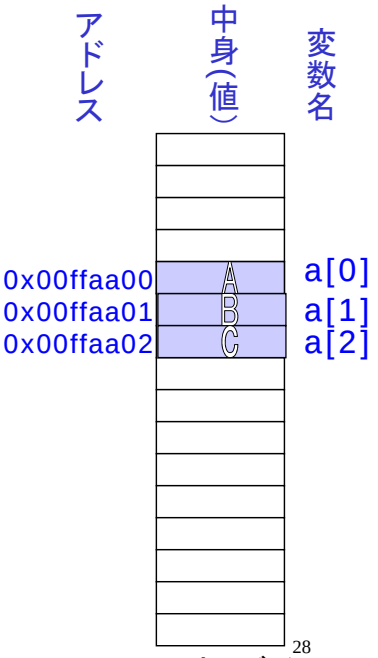


27

イメージ

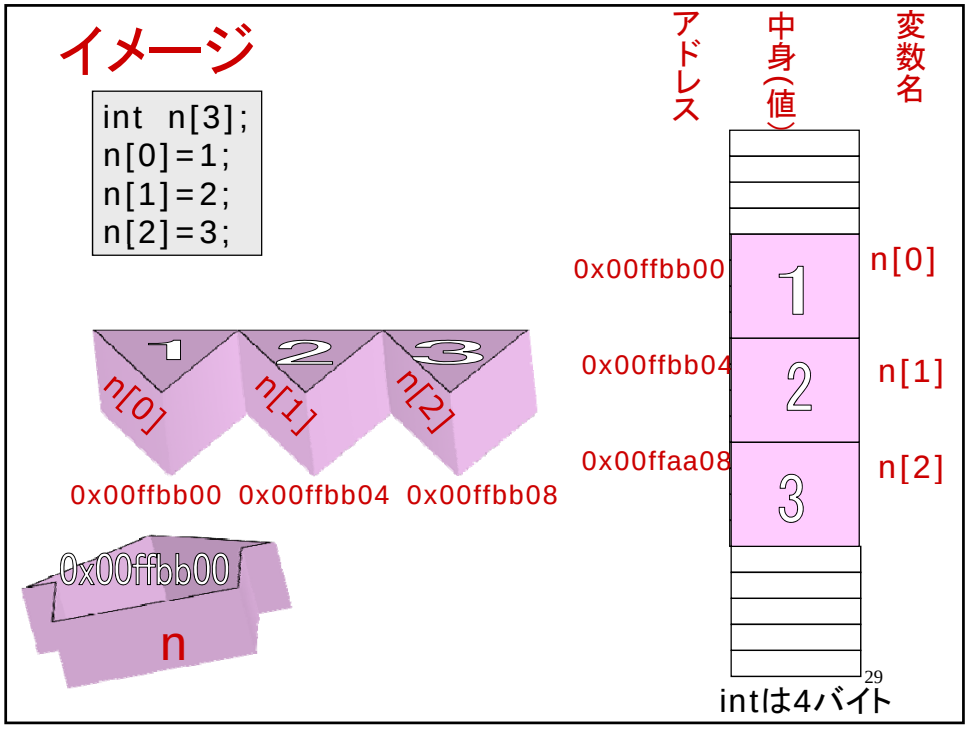
```
char a[3];
a[0]='A';
a[1]='B';
a[2]='C';
```

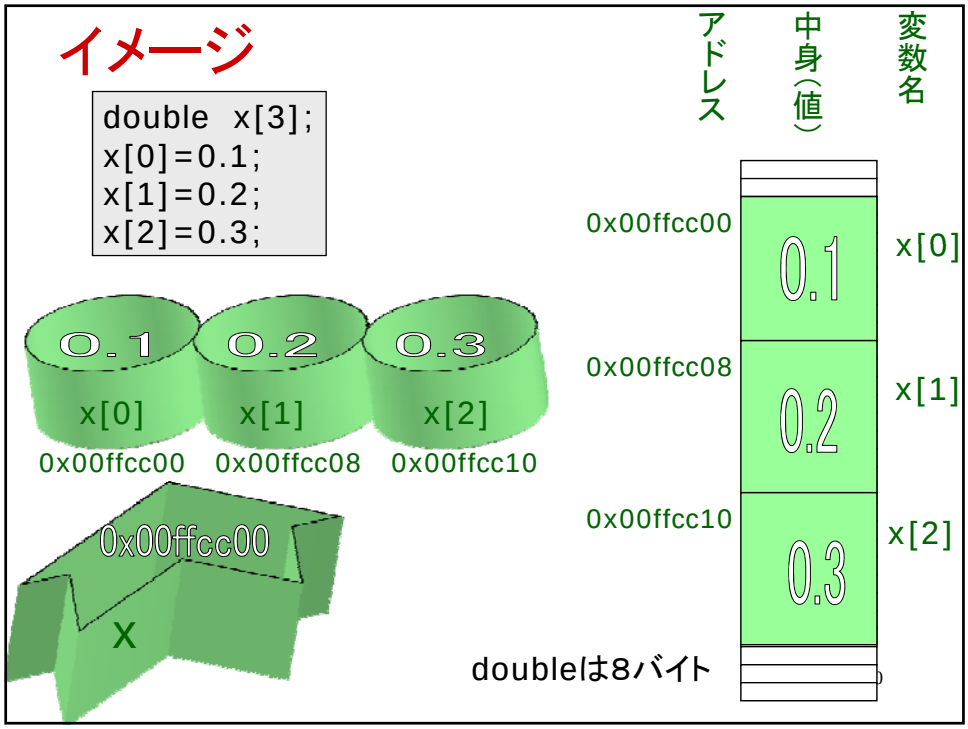




28

charは1バイト


イメージ
int n[3];
n[0]=1;
n[1]=2;
n[2]=3;
1
2
3
n[0]
n[1]
n[2]
0x00ffbb00 0x00ffbb04 0x00ffbb08
0x00ffbb00
n
アドレス
中身(値)
変数名
0x00ffbb00
0x00ffbb04
0x00ffaa08
n[0]
n[1]
n[2]
intは4バイト


イメージ
double x[3];
x[0]=0.1;
x[1]=0.2;
x[2]=0.3;
0.1
0.2
0.3
x[0]
x[1]
x[2]
0x00ffcc00 0x00ffcc08 0x00ffcc10
0x00ffcc00
x
アドレス
中身(値)
変数名
0x00ffcc00
0x00ffcc08
0x00ffcc10
x[0]
x[1]
x[2]
doubleは8バイト

練習３

```
/* pointer_array.c ポインターと配列実験 コメント省略    */
#include <stdio.h>
#define  MAX  5
int  main()
{
      int  i;
      int    n[MAX];     /*データを入れる配列*/
      int    *p;    /*上の配列の先頭を指すポインタ*/

      for(i=0;i< MAX;i++)
         {
            n[i]=i+1;
         }

      /*     次に続く       */
```



```
      /*     続き*/
      m_ptr=n;
      printf("nの値は %p  ¥n",n);
      printf("n[0]のアドレスは%p  ¥n",&n[0]);
      printf("m_ptrの値は%p  ¥n",p);
      printf("¥n¥n");

      printf("&n[i] n[i] p[i]¥n");
      for(i=0;i<MAX;i++)
         {
            printf("%p    %d    %d  ¥n",
            &n[i],n[i],p[i]);
         }
      return  0;
}
```

演算子＆と＊の結合力

演算子＆、＊の結合力は、算術演算子よりつよく、
インクリメント演算やデクリメント演算よりよわい。

++ --　＞　& *(単項演算子)　＞　/ *(2項演算子)　＞　+ –

*p++;	は	*(p++);	の意味
*p+1;		(*p)+1;	

1つの式内でインクリメント演算子と間接演算子を使うときには、
括弧を用いて意図を明確にすること。

33

関数とポインタ

2つの関数間の引数の受け渡しに、ポインタは重要な役割を果たす。
関数の仮引数や戻り値として、アドレスを用いることができる。

書式

```
void　　関数名(仮引数の型　＊　仮引数名)
{

　　return;
}
```

例

```
void　func_ref(int　*P)
{
　return;
}
```

```
int main()
{
　func_ref(&i);
　a=i;
}
```

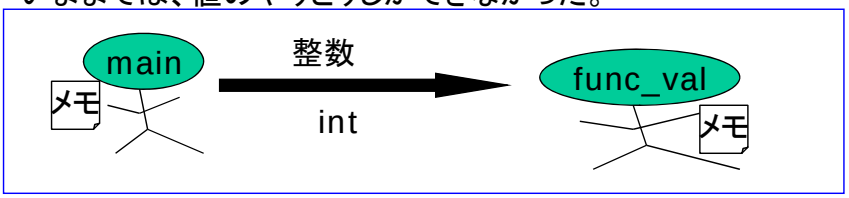
“ほぼ”同一の効果

```
int　func_val(int k)
{
　return k;
}
```

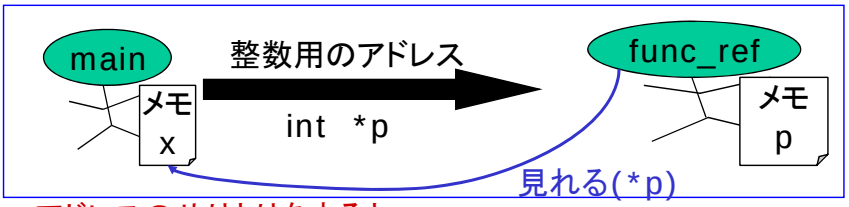
```
int main()
{
　i=func_val(i);
　a=i;
}
```

イメージ

いままでは、値のやりとりしかできなかった。



ポインタの仮引数を用いると、アドレスのやりとりができる。

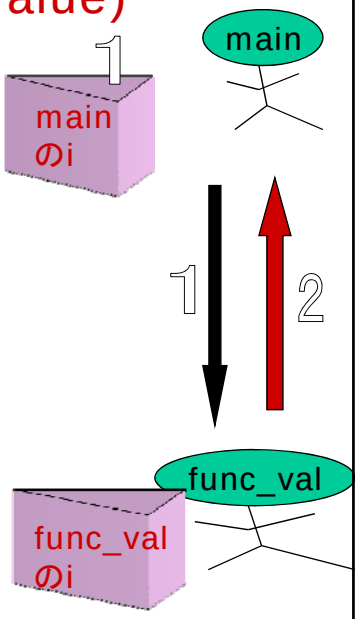


アドレスのやりとりをすると、
他の関数内のローカル変数の内容を変更できる。

35

値による呼び出し(call by value)

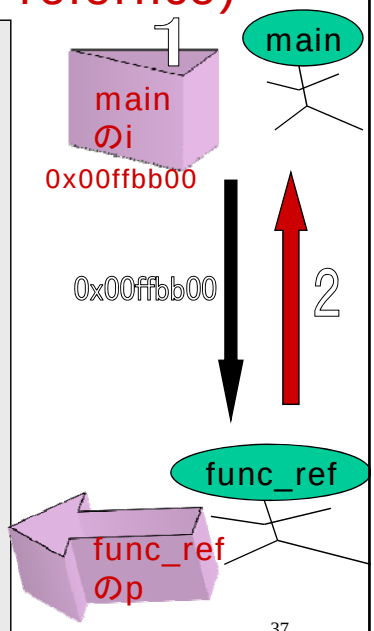
```
/*test_cbv.c*/
#include <stdio.h>
int  func_val(int);
int  main()
{
        int i;
        int j;
        i=1;
        j=0;
        printf("i=%d  j=%d¥n",i,j);
        j=func_val(i);
        printf("i=%d  j=%d¥n",i,j);
        return  0;
}
int  func_val(int i)
{
        i++;
        return i;
}
```



36

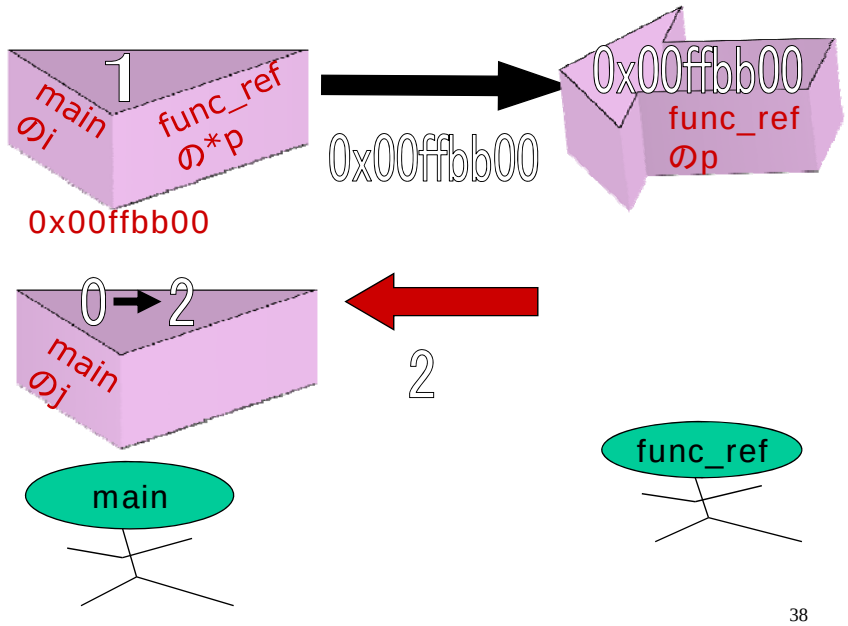
参照による呼び出し(call by refernce)

```
/*test_cbr.c*/
#include <stdio.h>
int func_ref(int *);
int  main()
{
        int i;
        int j;
        i=1;
        j=0;
        printf("i=%d j=%d¥n",i,j);
        j=func_ref(&i);
        printf("i=%d j=%d¥n",i,j);
        return 0;
}
int  func_ref(int   *p)
{
        (*p)++;
        return (*p);
}
```



37

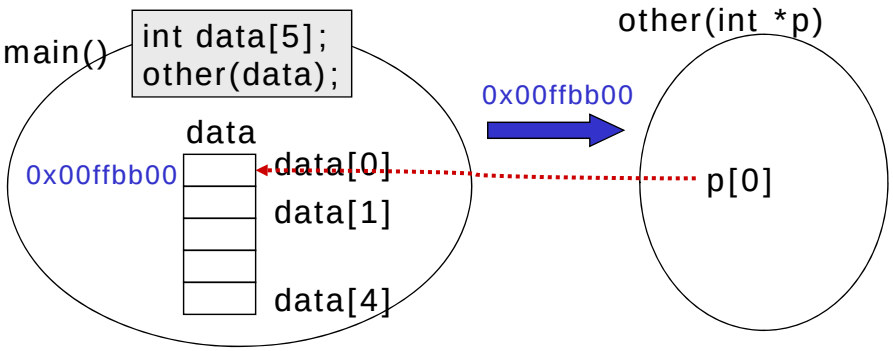
イメージ



38

関数間での配列の引き渡し1

他の関数に配列(data[**])の先頭要素のアドレス(data,すなわち、&data[0])を渡すことができる。配列名が配列の先頭アドレスを保持していることに注意する。

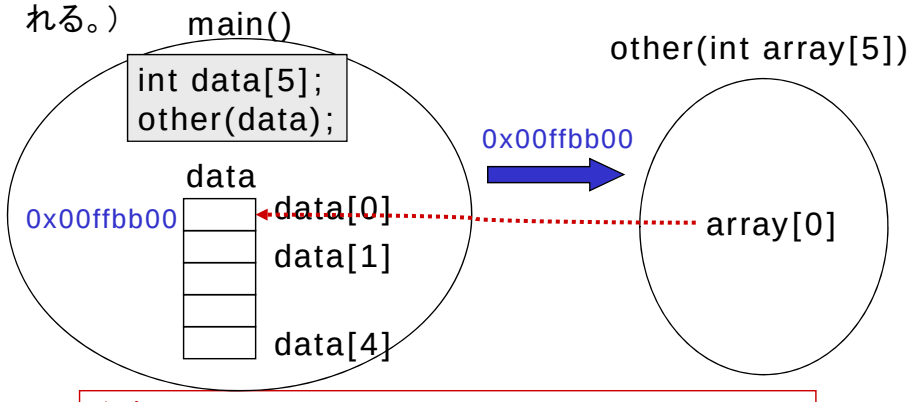


受け取った他の関数の中では、配列の先頭要素のアドレスの入ったポインタを配列名のように使うことができる。

39

関数間での配列の引き渡し2

受け取る側では、仮引数に配列を記述しても良い。
この場合、引き渡されたアドレスが、引数の配列要素の先頭アドレスになる。（すなわち、「array＝data」の代入が行なわれる。）



注意：
呼び出し側の配列の内容が書き換わるかもしれない。
十分に注意して関数を設計すること。

40

練習４

```
/*test_sendarray.c*/
#include <stdio.h>
#define   MAX  10
void    print_array(int n,int *p);
void    write_array(int n,int *p);
int  main()
{
        int  i;
        int  n;
        int  data[MAX];

        for(i=0;i<MAX;i++)
           {
               data[i]=i;
           }

        printf("n=?");
        scanf("%d",&n);
        /*      次に続く  */
```



```
        /*続き*/
        print_array(n,data);

        /*内容変更*/
        write_array(n,data);

        print_array(n,data);
        return 0;
}
/*main関数終了*/

        /*次に続く*/
```

```
/*      続き     */
void   print_array(int n,int  *p)
{
        int  i;
        for(i=0;i<n;i++)
           {
               printf("data[%d] =  %d ¥n",i,p[i]);
           }
        printf("¥n");
        return;
}

/*      次に続く         */
```

注意：
呼ばれる方では、配列の最後に気を付ける事。

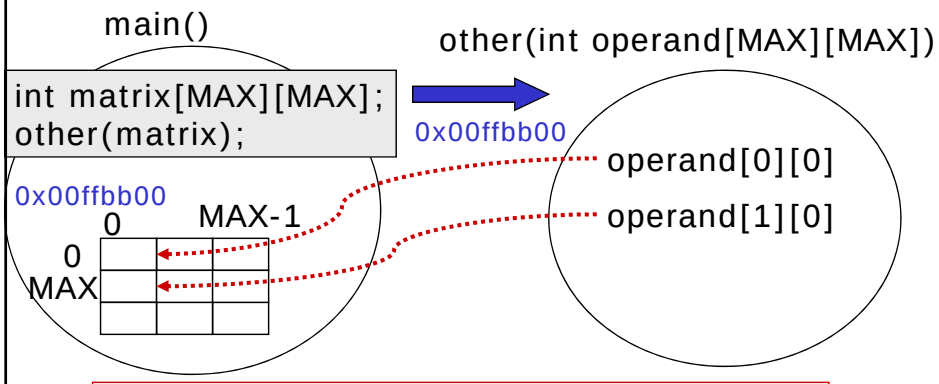
43

```
/*      続き     */
void   write_array(int n,int  *p)
{
        int  i;
        for(i=0;i<n;i++)
           {
               p[i]=(p[i])*2;
           }
        return;
}
/*全てのプログラム終了*/
```

44

関数間での2次元配列の引き渡し

2次元配列では、先頭アドレスの他に大きさも指定する必要がある。



注意:
呼び出し側の配列の内容が書き換わるかもしれない。
十分に注意して関数を設計すること。

45

他の関数内の2つの変数の値を交換する関数

```
/*
        作成日:yyyy/mm/dd
        作成者:本荘太郎
        学籍番号:B0zB0xx
        ソースファイル:pointer_swap.c
        実行ファイル:pointer_swap
        説明:参照呼び出しの効果を調べるための
                プログラム。ポインタを用いて、他の関数内
                の2つの変数の値を交換する関数を作成し
                利用する。
        入力:標準入力から2つの整数を入力。
        出力:標準出力にその2つの整数を交換して出力する。
/*      次のページに続く      */
```

46

```
/*     続き    */
/* ヘッダファイルの読み込み*/
#include <stdio.h>

/* マクロの定義 */
/* このプログラムでは、マクロは用いない。*/

/*グローバル変数の宣言*/
/*このプログラムでは、グローバル変数は用いない。*/

/* プロトタイプ宣言*/
void   swap(int  *p,int  *q);
       /*参照呼び出しを用いて、
          2つの変数の値を入れ替える関数*/
/*     次のページに続く     */
```

47

```
/*     続き    */
/*main関数*/
int    main()
{
       /*ローカル変数宣言*/
       int    data1;       /*入力整数1*/
       int    data2;       /*入力整数2*/


       /* 入力処理*/
       printf("data1=?");
       scanf("%d",&data1);
       printf("data2=?");
       scanf("%d",&data2);
```

46

```
        /*      続き    */

        /*2つの変数の値の交換 */
        swap(&data1,&data2);  /*関数swapの引数1、引数2
                                  とも参照呼出しなので、
                                  アドレスを与える。*/
        /*データ出力*/
        printf("data1= %d¥n",data1);
        printf("data2= %d¥n",data2);

        return 0;    /*正常終了*/
}
/*main関数終了*/

/*      次に続く        */
```



```
/*他の関数の2つの変数の値を交換する関数
仮引数 p,q:交換すべき整数型の変数のアドレス
    （参照呼出なので、呼び出し側ではアドレスを指定する）
戻り値:なし
*/
void   swap(int *p, int *q)
{
        /*ローカル変数宣言*/
        int temp;    /*2つの変数の値を交換するために、
                        値を一時的に蓄えて おく変数。*/

        /*処理内容の通知*/
        printf("交換中 ¥n");

        /*      次に続く        */
```

```
        /*     続き    */

        /*     命令記述      */
        temp=(*p);
        (*p)=(*q);
        (*q)=temp;

        return;
}
/*swap関数終了*/
/*全プログラム(pointer_swap.c)終了*/
```


実行例

```
$./pointer_swap
data1=? 1
data2=? 3
交換中
data1= 3
data2=1
$
```