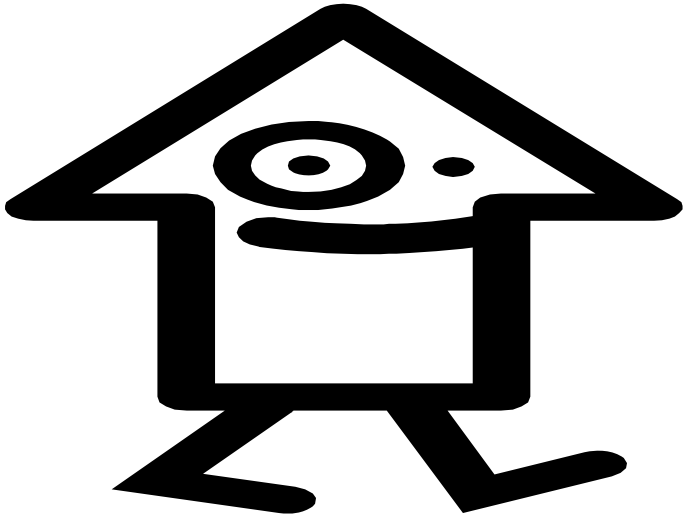
第１１回ポインタの基礎
（アドレスとポインタ）



今回の目標

- C言語におけるポインタを理解する。
- 変数のアドレスを理解する。
- ポインタ型を理解する。
- アドレス演算子、参照演算子の効果を理解する。
- NULLというアドレスを理解する。

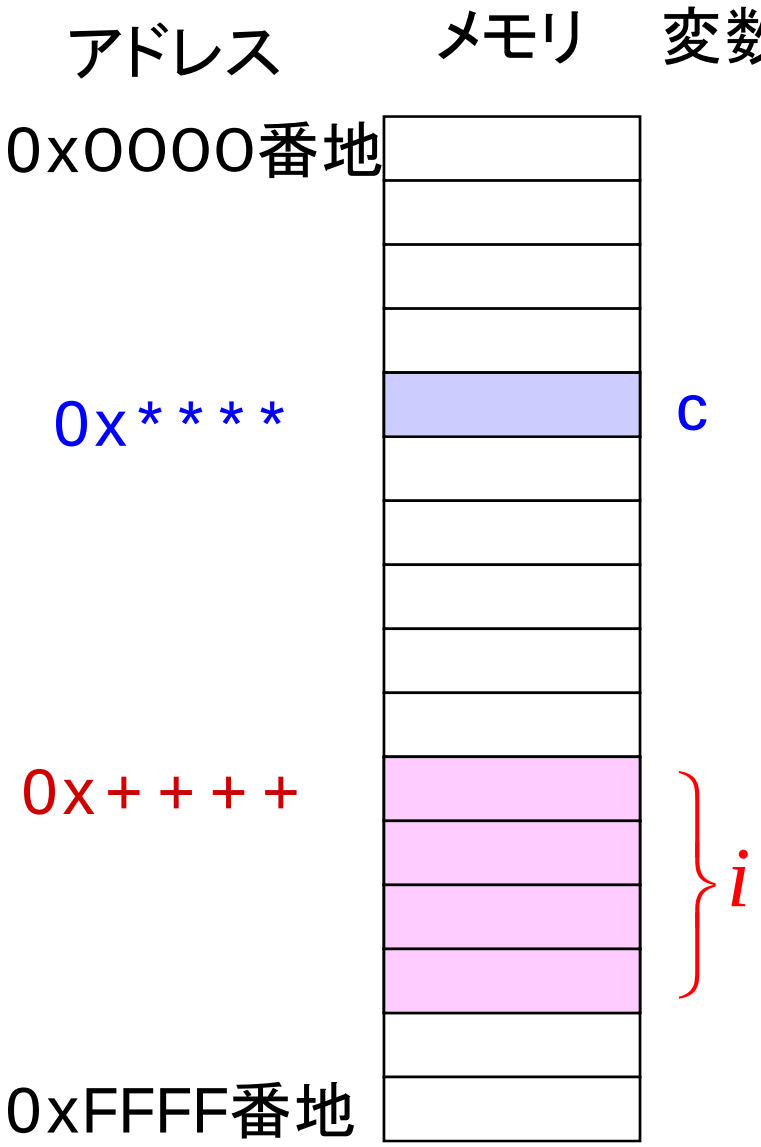
☆複数の関数内で変数のアドレスを表示するプログラムを作成する。

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。

ポインタの型は
これまでのどの型（char,int,double)とも異なる。

変数とアドレス



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。

アドレス演算子　　&

&：変数に割り当てられたメモリの先頭アドレスを求める演算子。
前置の単項演算子。

書式

```
&　変数名
```

例

```
int　age;
scanf("%d",&age);
```

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

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。

アドレスを指定するscanf文の仕様

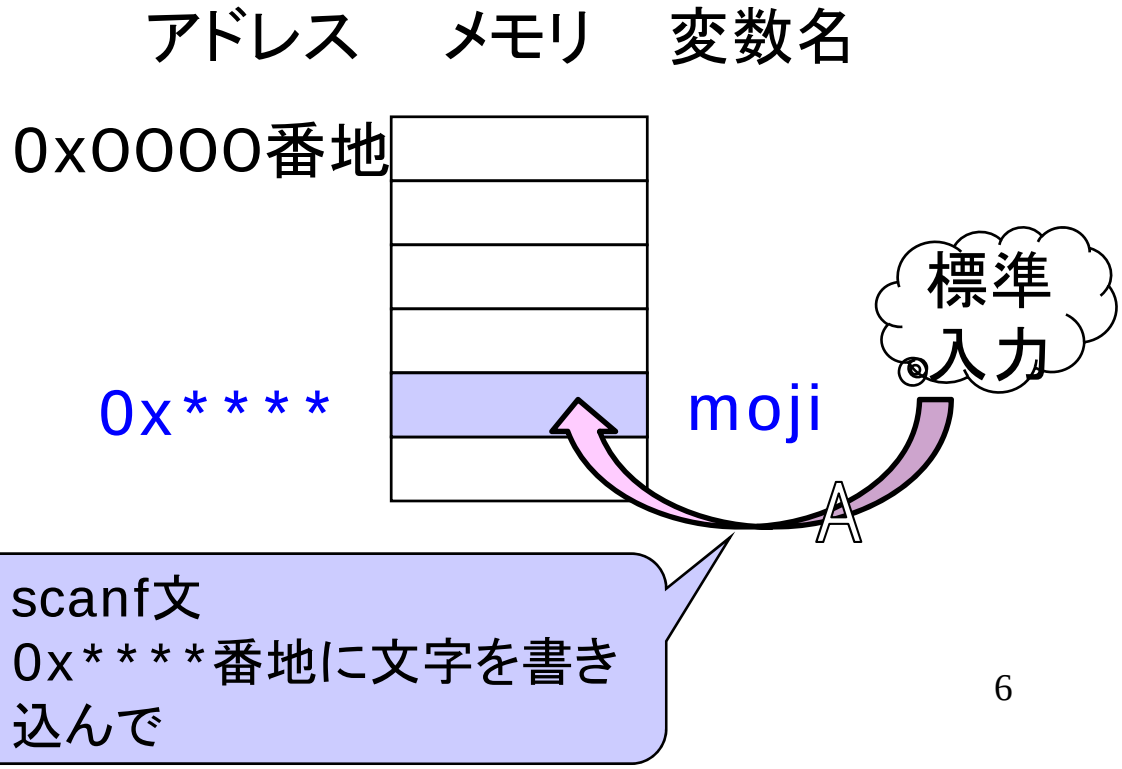
書式

```
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。



アドレスを表示するprintf文の仕様

printf文には、アドレスを表示するための変換仕様がある。

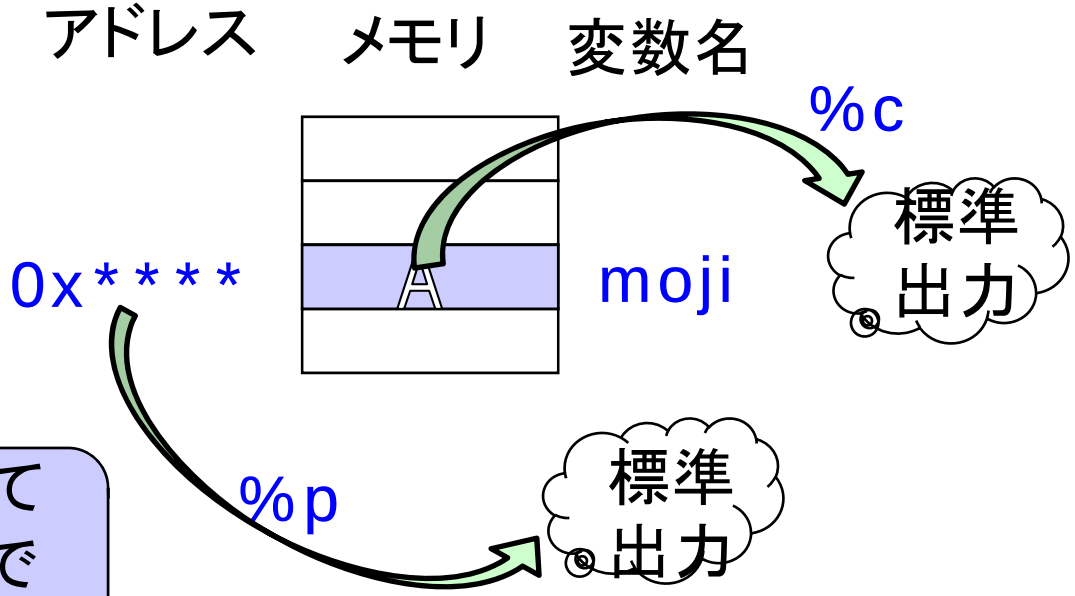
例　　%p ⟷ アドレス
(型の区別無し)

変数のアドレスを
表示させる書式

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

&がついているので
アドレス

16進数で表示される。



変数の中身(値)
を表示させる書式

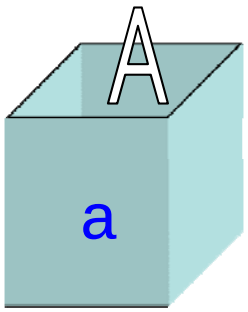
```
char　moji;
printf("%c",moji);
```

&がついて
いないので
中身(値)

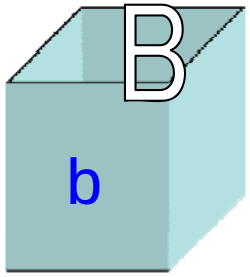
文字として
表示される。

イメージ

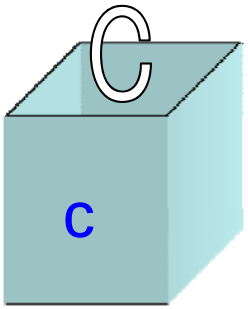
0x****
番地



0x++++
番地



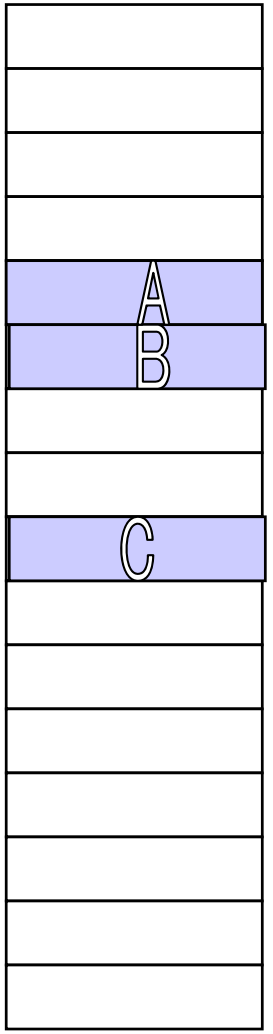
0x####
番地



アドレス　中身（値）　変数名

0x****　a
0x++++　b

0x####　c

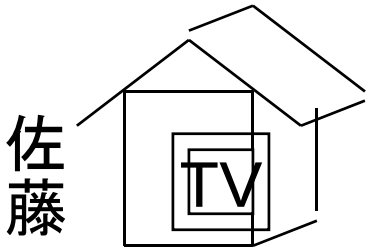


char型は1バイト

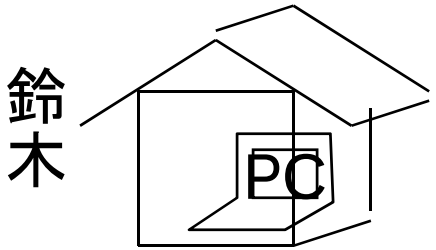
イメージ

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

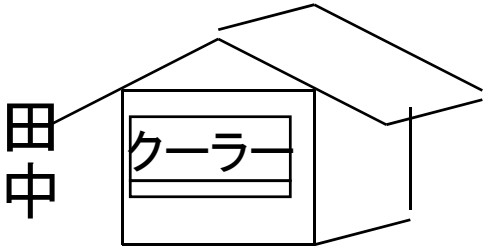
変数における
変数名と
アドレスと
中に入っている値



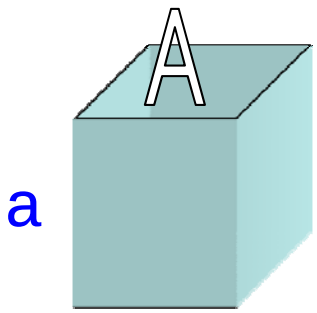
1-23-4



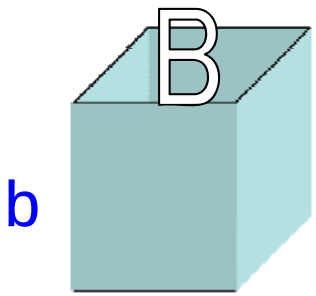
1-23-5



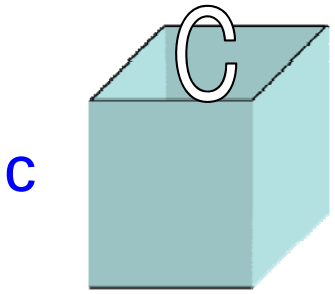
1-23-6



0x****番地



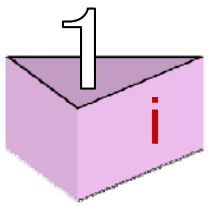
0x++++番地



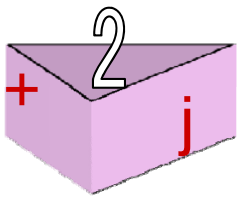
0x####番地

イメージ

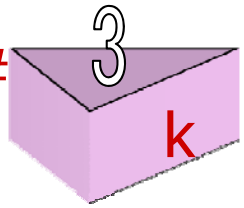
0x****
番地

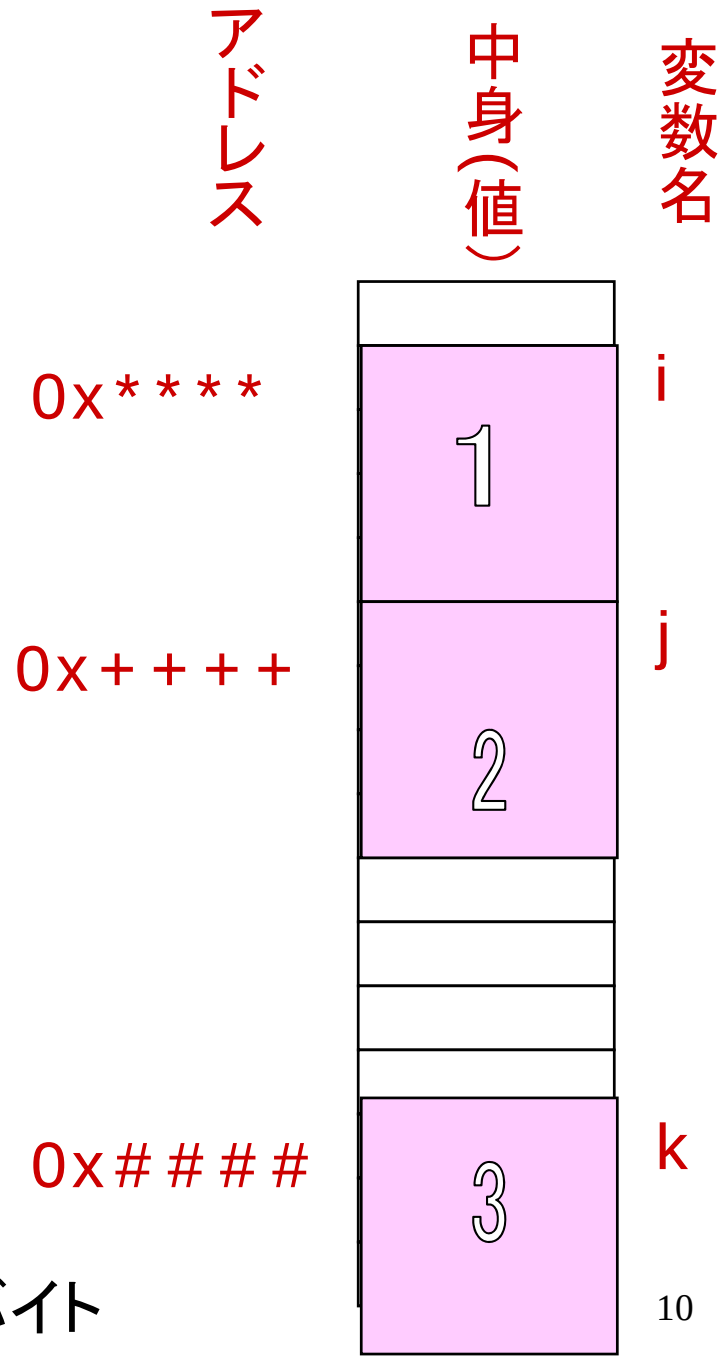


0x++++
番地



0x####
番地

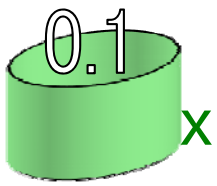




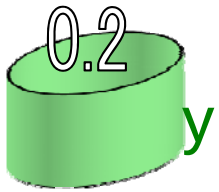
int型は4バイト

イメージ

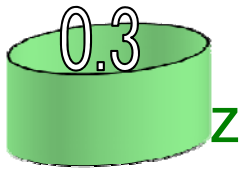
0x****
番地

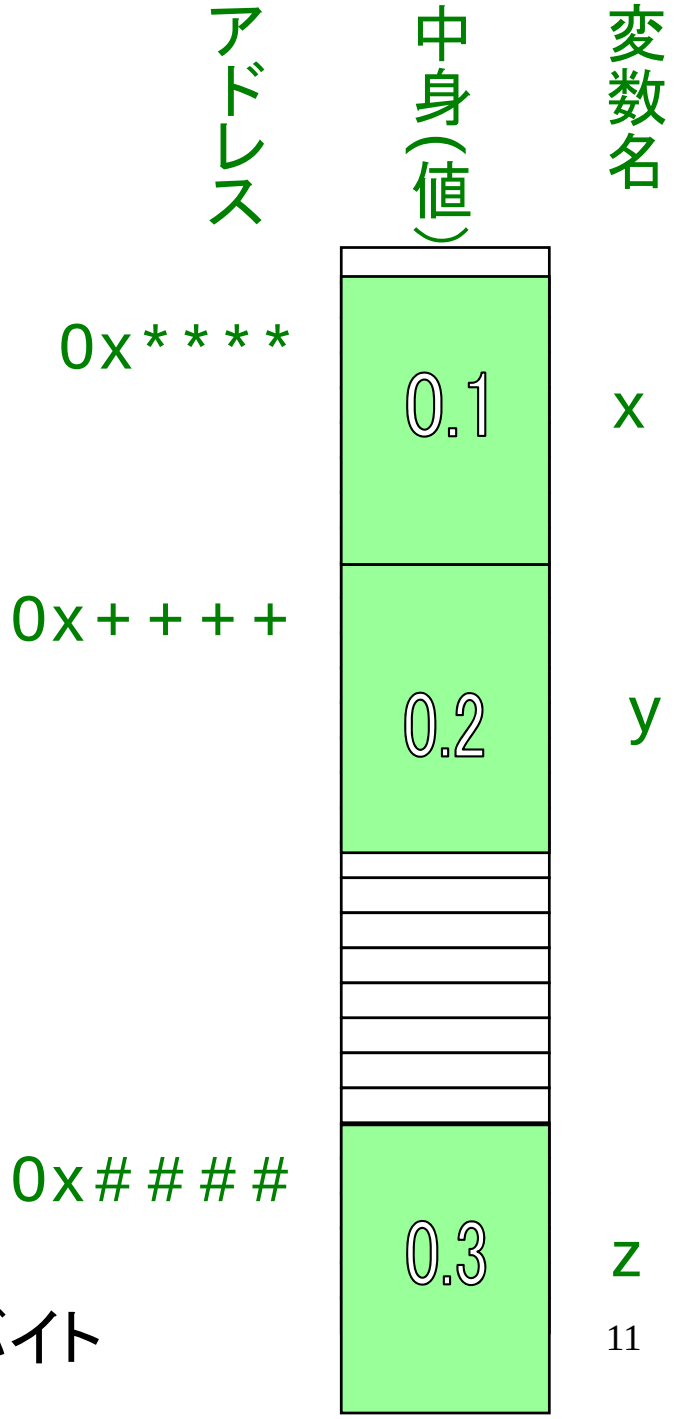


0x++++
番地



0x####
番地





double型は8バイト

練習1

```
/* 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

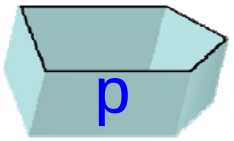
```
データ型　　＊ポインタの名前；
```

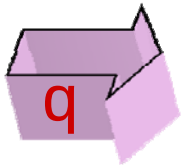
例

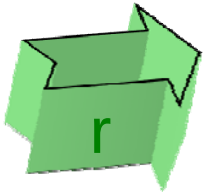
```
char　*p;
```

```
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）

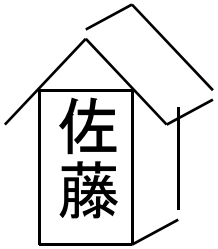
pは（char　*）型の変数と考えてもよい。
char型と（char　*）型は異なる型。

イメージ

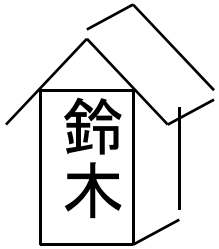
（種類別の）建物
住所
（種類別の）アドレス帳

対応

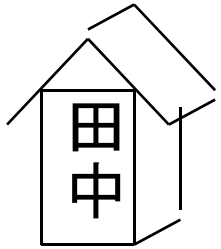
変数（の型）
アドレス
（ある型の変数を指す）ポインタ



1-23-4

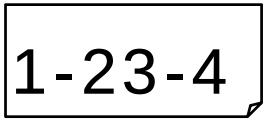


1-23-5



1-23-6

民家専用の
アドレス帳

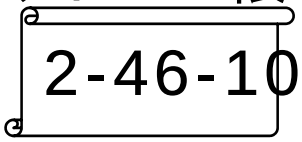


2-46-8

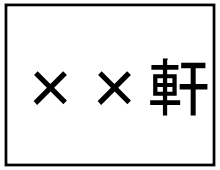
2-46-9

2-46-10

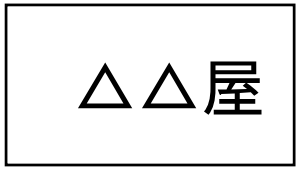
工場専用の
アドレス帳



3-6-9

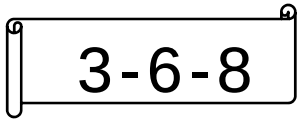


3-6-8



3-6-7

店専用の
アドレス帳



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。

書式

＊ポインタ

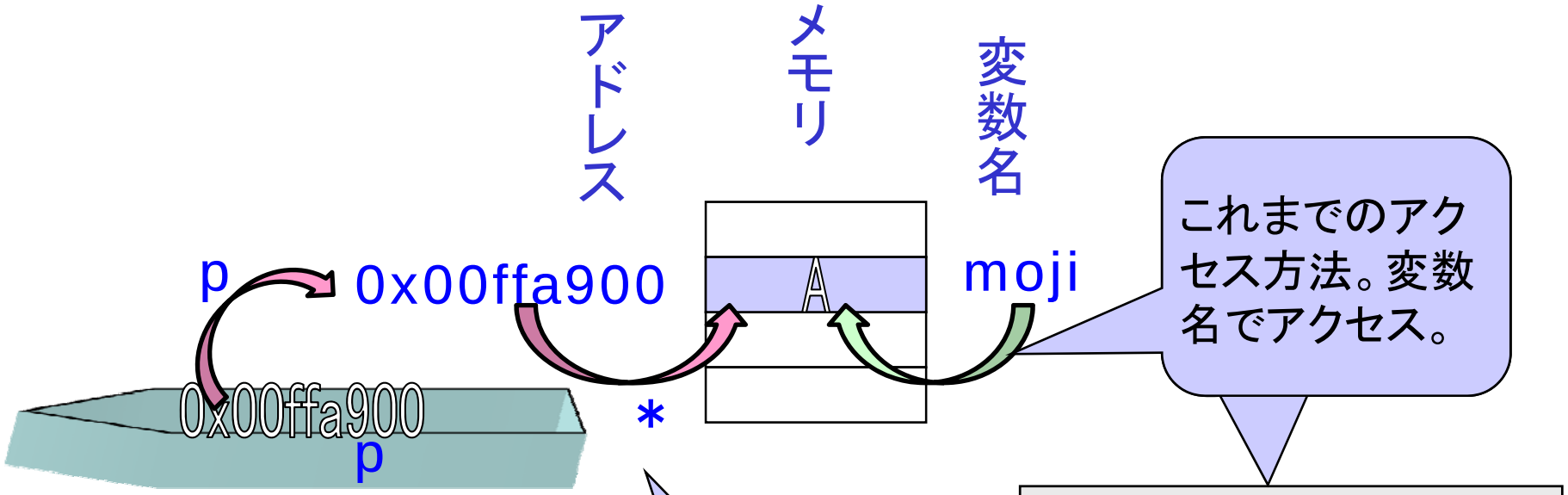
＊：ポインタに格納されているアドレスに割り当てられている変数を求める演算子。
前置の単項演算子

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。

間接演算子を用いたメモリアクセス



これまでのアクセス方法。変数名でアクセス。

```
char moji;

printf(“%c”,moji);
```

```
char moji;
char *p;

p=&moji;

printf(“%c”,*p);
```

ポインタを用いたアクセス方法。ポインタ名と間接演算子を用いてアクセスする。アドレスの役割に注意する。

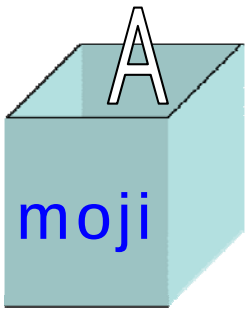
ポインタによる変数の別名

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。

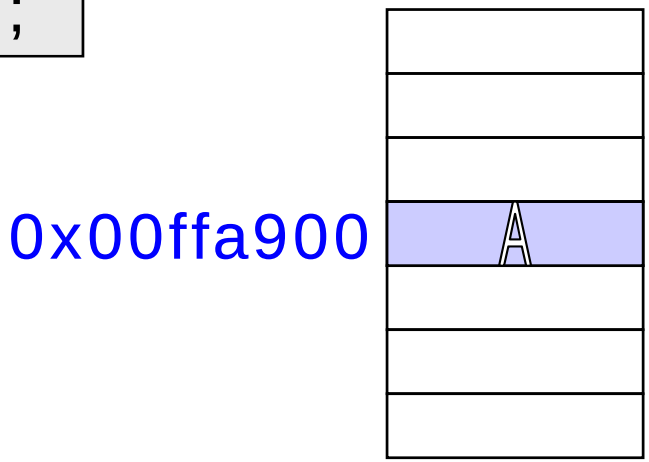
```
char  a;
char  b;
char  *p;

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。*/
```

イメージ

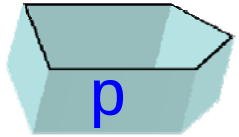
```
char  a;
moji='A';
```



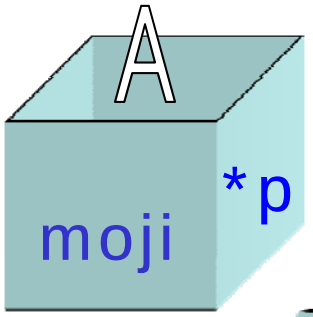
0x00ffa900




```
char  *p;
```




```
p=(&moji);
```



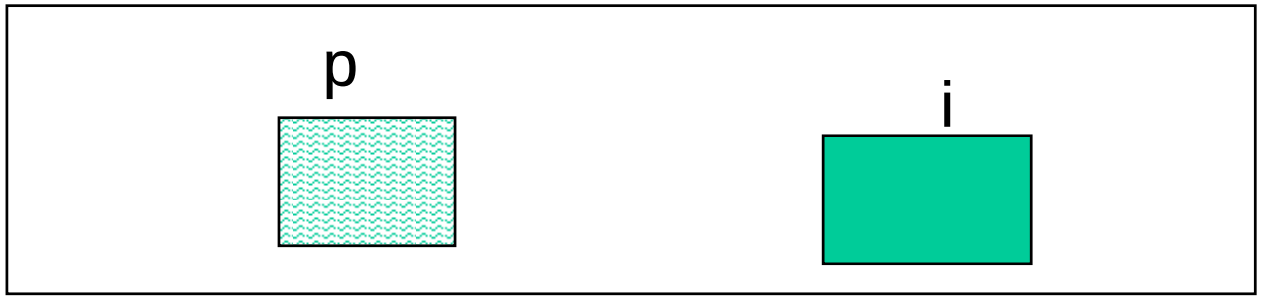
0x00ffa900

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。

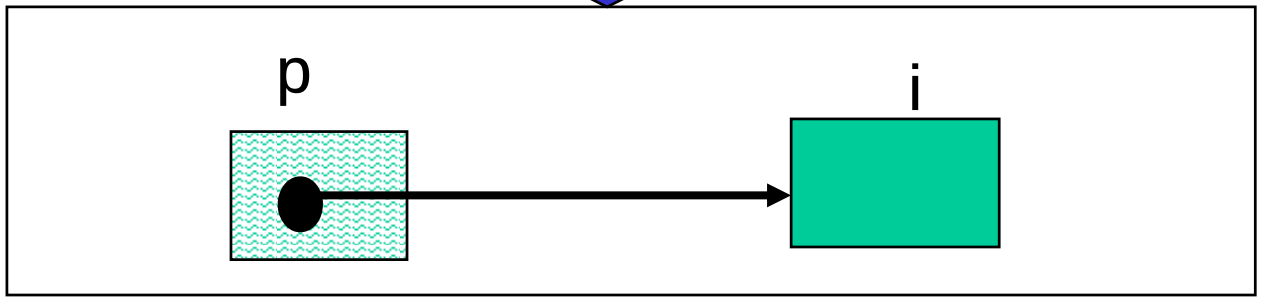
変数宣言（変数の用意）

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



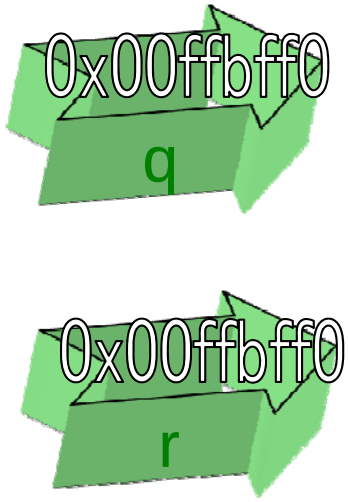
アドレス代入

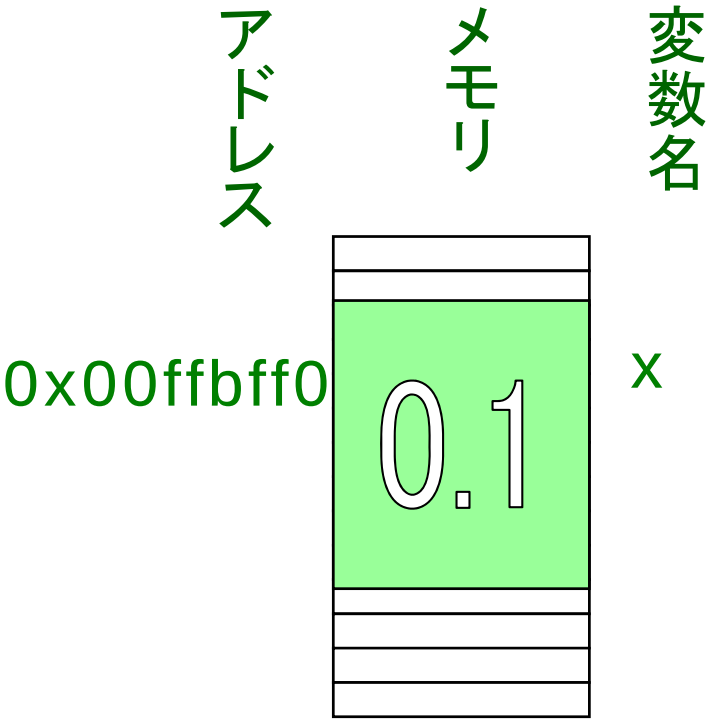
```
p=(&i);
```

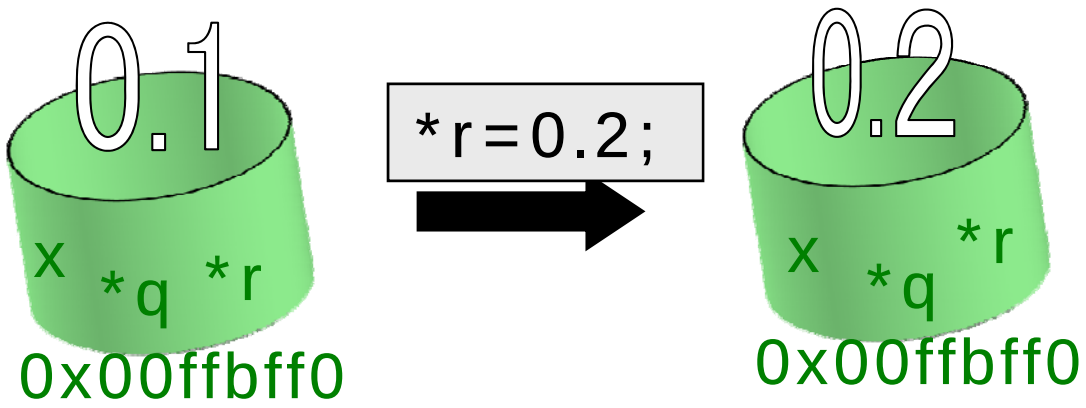


複数の別名

```
double  x=0.1;
double *q;
double *r;
q=&x;
r=&x;
*r=0.2;
```







NULLというアドレス

どの変数も指さない特別なアドレスを
NULL
として表す。

```
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
できる。

×

```
int      *p;
p=NULL;
/*次は間違い*/
*p=1;
```

NULLを格納するポインタ
へは間接演算子を用いる
ことはできない。

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        */
```

```
/*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*/
```

```
        /*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;
}
```

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。

++
--
> &(アドレス演算子)
＊(間接演算子)
> /
＊（算術演算子）
> +
—

```
*p++;
```

は

```
*(p++);
```

```
(*p)+1;
```

の意味

```
*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。

各種変数のアドレスを表示するプログラム

```
/*
        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：print_address.c
        実行ファイル：print_address
        説明：変数とアドレスの関係を理解するためのプログラム。
              複数の関数内でアドレスを表示する。
        入力：標準入力から2つの整数値を入力する。
        出力：標準出力に以下を出力する。
              main関数ローカル変数のアドレスと値，
              main関数以外のローカル変数のアドレスと値、
              仮引数のアドレスと値。
            /* 次のページに続く      */
```

```
/*      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*/
/*変数のアドレスを表示する関数*/
void function(int data1);
/*      次のページに続く      */
```

```
/*      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*/
        int     *p;             /*ポインタ*/

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

```
        /*      続き    ポインタの設定*/
        p=(&data2);

        /*関数呼び出し前*/
        /*変数の中身と変数アドレスの表示*/
        printf(“main関数内での表示¥n”);
        printf(“関数function呼び出し前¥n”);
        printf(“&data1=%10p”,&data1);
        printf(“  data1=%2d¥n”,data1);
        printf(“&data2=%10p”,&data2);
        printf(“  data2=%2d¥n”,data2);
        printf(“      &p=%10p”,&p);
        printf(“   p=%10p”,p);
        printf(“ *p=%2d¥n¥n”,*p);

        /*関数呼び出し*/
         function(data1);
/* 次に続く   */
```

```
        /*      続き    */
        /*関数呼び出し後*/
        /*変数の中身と変数アドレスの表示*/
        printf(“main関数内での表示¥n”);
        printf(“関数function呼び出し後¥n”);
        printf(“&data1=%10p”,&data1);
        printf(“  data1=%2d¥n”,data1);
        printf(“&data2=%10p”,&data2);
        printf(“  data2=%2d¥n”,data2);
        printf(“      &p=%10p”,&p);
        printf(“   p=%10p”,p);
        printf(“ *p=%2d¥n¥n”,*p);

        return 0;    /*正常終了*/
}
/*main関数終了 次に続く */
```

```
/*ローカル変数の値とローカル変数のアドレスを表示する関数
仮引数 data1:main関数から値を受け取る。仮引数のアドレスを
                調べるために用意してある。
戻り値:なし
*/
void   function(int data1)
{
        /*ローカル変数宣言*/
        int data2;    /*整数型の変数。main関数内にある変数
                        と同じ名前にしてある。*/
       int * p;       /*ポインタ。main関数内にあるポインタ
                        と同じ名前にしてある。*/

        /*処理内容の通知*/
        printf(“function 実行中¥n");

        /*      次に続く        */
```

```
        /*      続き    ポインタの設定*/
        p=(&data2);

        /*変数の中身と変数アドレスの表示*/
        printf("関数function内での表示¥n");
        printf("&data1=%10p",&data1);
        printf("  data1=%2d¥n",data1);
        printf("&data2=%10p",&data2);
        printf("  data2=%2d¥n",data2);
        printf("      &p=%10p",&p);
        printf("   p=%10p",p);
        printf(" *p=%2d¥n¥n",*p);

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

実行例

```
$./print_address < print_address.in
main関数内での表示
関数function呼び出し前
&data1=0xbfba3160  data1= 3
&data2=0xbfba315c  data2= 4
     &p=0xbfba3158 p=0xbfba315c  *p=4

function 実行中
関数function内での表示
&data1=0xbfba3140  data1= 3
&data2=0xbfba3134  data2=-1208557580
     &p=0xbfba3130  p=0xbfba3134  *p =-1208557580

main関数内での表示
関数function呼び出し後
&data1=0xbfba3160  data1= 3
&data2=0xbfba315c  data2= 4
     &p=0xbfba3158 p=0xbfba315c  *p=4

$
```