

# データ構造入門及び演習

## 3回目:ポインタ1

### ~基本的な仕組み~

---

2014/04/25

担当:見越 大樹

61号館304号室

# アドレス

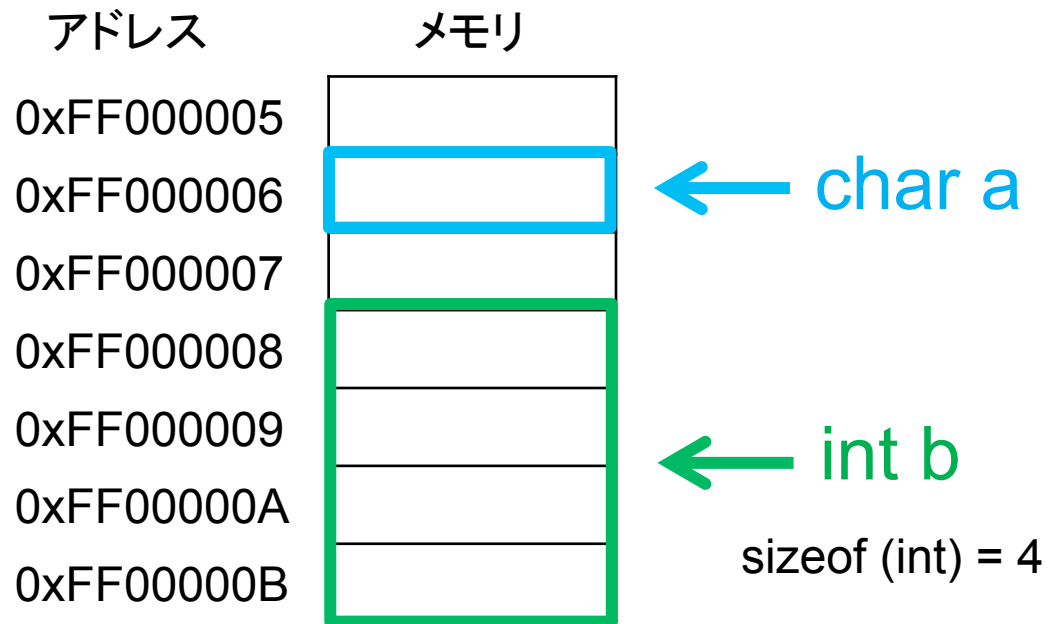
- 変数や配列はコンピュータのメモリ上にある
- 変数や配列の値はメモリ上に記録されている
- メモリには「アドレス」という連続した番号がついていて、どこに何が入っているかを管理できる

アドレスの表し方:

変数名の頭に&をつける

&a = 0xFF000006

&b = 0xFF000008



1つのブロックで1バイト

# アドレス表示のサンプルプログラム

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

```
void main()
```

```
{
```

```
    char a='T';    int b=55;
```

```
    printf("aは「%c」です. \n", a);
```

```
    printf("aのアドレスは「%p」です. \n", &a);
```

```
    printf("bは「%d」です. \n", b);
```

```
    printf("bのアドレスは「%p」です. \n", &b);
```

```
    printf("\n");
```

```
}
```

```
aは「T」です.  
aのアドレスは「0x2abd0f」です.  
bは「55」です.  
bのアドレスは「0x2abd08」です.
```

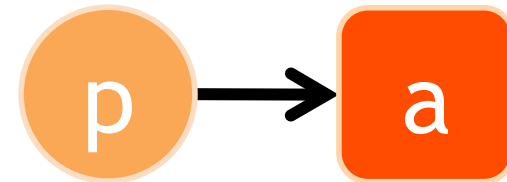
実行結果

アドレスを表示するときは%p

# ポインタ

- ポインタとは？
  - 変数が格納されている位置(アドレス)を値とする変数
- ポインタの宣言：
  - 型名の後ろ もしくは 変数名の前にアスタリスクをつける  
char\* p; または char \*p;  
(pをポインタ変数と呼ぶ)
- ポインタへのアドレスの代入：
  - ポインタpに変数aのアドレスを代入

```
char* p;  
char a='T';  
p = &a; (意味：p=aのアドレス)
```

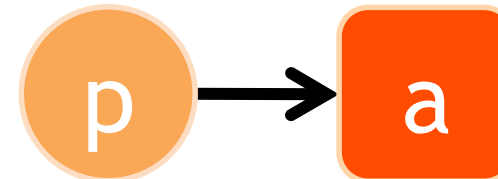


pはaを指す  
(pはaのアドレスを持つ)

# ポインタが指す値の参照

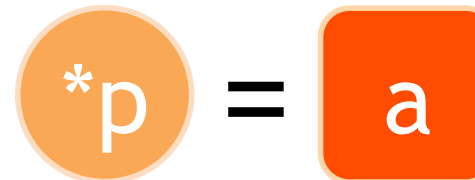
- ポインタ変数の前に「\*(アスタリスク)」をつけると、そのポインタが指す先の値を参照できる

```
char* p;  
char a='T';  
p = &a; (意味：p=aのアドレス)
```



pはaを指す  
(pはaのアドレスを持つ)



```
char b;  
b = *p; (意味：b='T')
```




\*pはaと同じ意味を持つ  
\*pはaのエイリアスと呼ぶ

注意：b=\*pの「\*」と、char\* pの「\*」は意味が異なる！

# 変数の型と格納できる値

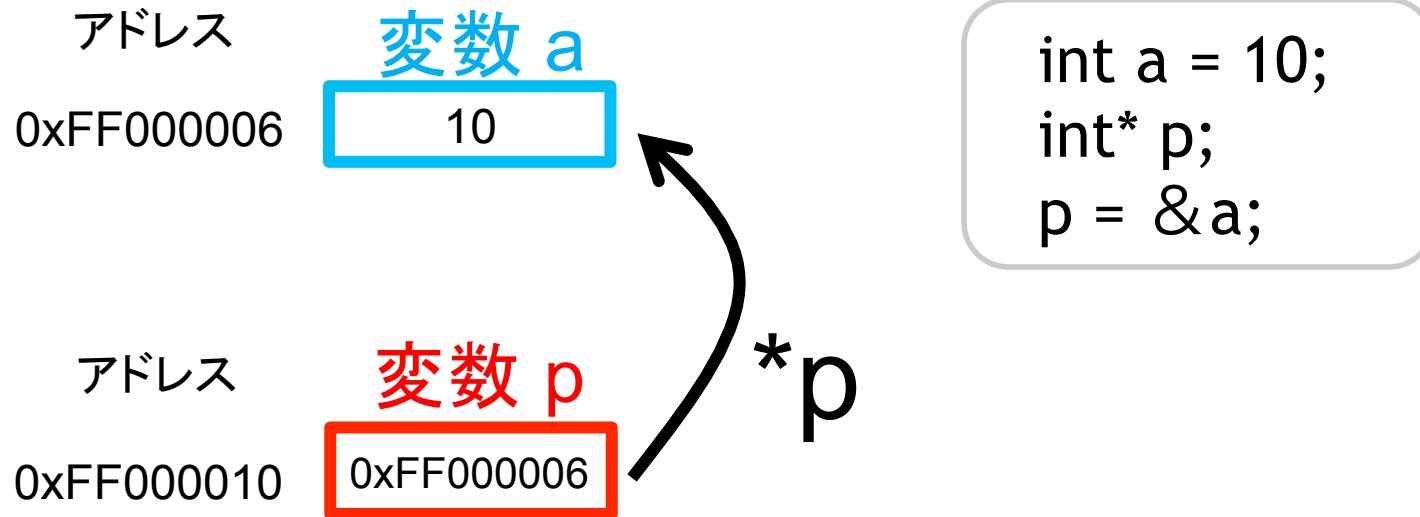
アドレス	メモリ		
0xFF000006		← int a	変数aに整数型を格納できる
0xFF000010		← int* p	変数pにint 型変数のアドレスを格納できる

---

アドレス	メモリ	
0xFF000006		← int a
0xFF000010		← int* p

```
int a = 10;  
int* p;  
p = &a;
```

# ポインタの値とエイリアス



「p」にはアドレスが入っている

「\*p」は変数pの指した先(アドレス)に飛ぶ  
(\*pとaは同じ意味を持つ)

# ポインタを使ったサンプルプログラム

```
#include <stdio.h>
void main(void){
    int a = 10;
    int* p;
    int b;

    p = &a; //アドレスの代入

    printf("変数aのアドレスは「%p」です\n", &p);
    printf("変数aの中身は「%d」です\n", a);
    printf("変数pの中身は「%p」です\n", p);
    printf("変数pの指す先の値は「%d」です\n", *p);
    printf("変数pのアドレスは「%p」です\n", &p);

    b = *p; // 変数bにpの指す先の値を代入
    printf("変数bのアドレスは「%p」です\n", &b);
    printf("変数bの中身は「%d」です\n", b);
}
```

変数aのアドレスは「0x8e41b690」です  
変数aの中身は「10」です  
変数pの中身は「0x8e41b690」です  
変数pの指す先の値は「10」です  
変数pのアドレスは「0x8e41b69c」です  
変数bのアドレスは「0x8e41b68c」です  
変数bの中身は「10」です

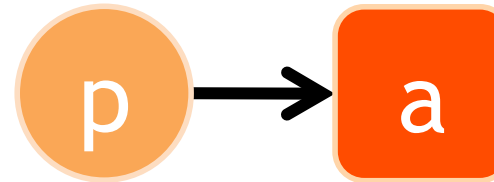


# ポインタのまとめ

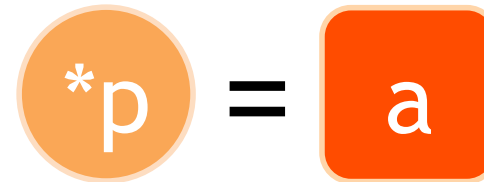
```
int a = 10;  
int b;  
int* p;  
p = &a;  
b = *p + 100;
```



```
p = &a;    //&はアドレス演算子  
b = *p + 20;  //*は関節演算子
```



pはaを指す  
(pはaのアドレスを持つ)



\*pはaと同じ意味を持つ  
\*pはaのエイリアスと呼ぶ

# ポインタの概念

## 「C言語最大の難関であり、最大の価値」

- ポインタ(pointer): 指し示すもの
  - C言語プログラムでは, データの記憶されている場所(アドレス)を指す変数
- ポインタを用いた処理:
  - アドレスを使用した処理のこと
  - 変数が入力されているメモリ上の場所を使用した処理
- 効果:
  - ポインタを使用しなくても大部分のプログラムは作成可能
  - C言語系特有のフレキシブルな記述であり, ポインタを使用することによって効率的にプログラミング可能である

# 変数のメモリ内配置

- プログラムに書かれた変数はメモリ内に自動的に配置される

```
#include <stdio.h>
int main(void)
{
    int nx = 2;
    char c='c';
    double dx = 3.14;
    int xc[3] = {5, 6, 7};
    ...
    return (0);
}
```

char 型変数 :	1 Byte
int 型変数 :	4 Byte
float 型変数 :	4 Byte
double 型変数 :	8 Byte
pointer は型によらず :	4 Byte

(32bitシステムの場合, 1Byte=8bit)

変数名	メモリ内容	アドレス
xc[0]	5	ffbefa40
xc[1]	6	ffbefa44
xc[2]	7	ffbefa48
dx	3.14	ffbefa50
c	'c'	ffbefa5a
nx	2	ffbefa5c

# ポインタ変数のサイズの確認

```
int main(void)
{
    char  a, *pa; //char a; char* pa;と同じ
    int    b, *pb;
    float  c, *pc;
    double d, *pd;

    printf("size of a  = %d byte\n", sizeof(a));
    printf("size of pa = %d byte\n\n", sizeof(pa));
    printf("size of b  = %d byte\n", sizeof(b));
    printf("size of pb = %d byte\n\n", sizeof(pb));
    printf("size of c  = %d byte\n", sizeof(c));
    printf("size of pc = %d byte\n\n", sizeof(pc));
    printf("size of d  = %d byte\n", sizeof(d));
    printf("size of pd = %d byte\n\n", sizeof(pd));

    return 0;
}
```

練習問題：実行結果を示しなさい。

# ポインタ変数のサイズの確認

```
int main(void)
{
    char  a, *pa; //char a; char* pa;と同じ
    int   b, *pb;
    float c, *pc;
    double d, *pd;

    printf("size of a = %d byte\n", sizeof(a));
    printf("size of pa = %d byte\n\n", sizeof(pa));
    printf("size of b = %d byte\n", sizeof(b));
    printf("size of pb = %d byte\n\n", sizeof(pb));
    printf("size of c = %d byte\n", sizeof(c));
    printf("size of pc = %d byte\n\n", sizeof(pc));
    printf("size of d = %d byte\n", sizeof(d));
    printf("size of pd = %d byte\n\n", sizeof(pd));

    return 0;
}
```

size of a = 1 byte  
size of pa = 4 byte

size of b = 4 byte  
size of pb = 4 byte

size of c = 4 byte  
size of pc = 4 byte

size of d = 8 byte  
size of pd = 4 byte

練習問題：実行結果を示しなさい。

# メモリアドレスの取得と表示

- `printf(“変数xのアドレス = %p\n” , &x);`
- 変数名からメモリアドレスを取得するには:
  - `&`: メモリアドレスを取得するための演算子  
(変数名の前に付ける)
  - `%p`: メモリアドレスと解釈して表示せよという指示

# アドレス演算子と間接演算子の使用例

- メモリ上アドレスの例

アドレス	変数名	内容
0番地	x0	0
4番地	x1	10
8番地	x2	20
12番地	x3	30
:	:	:
400番地	x100	1000

&x3 →

& アドレスを取り出す

\*(&x2) →

\* アドレスに格納されている値(内容)を示す

# アドレス演算子と間接演算子の使用例

- メモリ上アドレスの例

アドレス	変数名	内容
0番地	x0	0
4番地	x1	10
8番地	x2	20
12番地	x3	30
:	:	:
400番地	x100	1000

&x3 →

12番地

& アドレスを取り出す

\*(&x2) →

20

\* アドレスに格納されている値(内容)を示す



# ポインタを使用したプログラム例1

```
#include <stdio.h>
int main(void)
{
    int x1=10;
    int x2=20;
    printf("%d %p %d\n", x1, &x1, *(&x1));
    printf("%d %p %d\n", x2, &x2, *(&x2));
    return 0;
}
```

アドレス	変数名	内容
0番地	x0	0
4番地	x1	10
8番地	x2	20
12番地	x3	30
:	:	:
400番地	x100	1000

- 練習問題：実行結果を示しなさい

# ポインタを使用したプログラム例1

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

```
int main(void)
```

```
{
```

```
    int x1=10;
```

```
    int x2=20;
```

```
    printf(“%d %p %d\n”, x1, &x1, *(&x1));
```

```
    printf(“%d %p %d\n”, x2, &x2, *(&x2));
```

```
    return 0;
```

```
}
```

- 練習問題：実行結果を示しなさい

アドレス	変数名	内容
0番地	x0	0
4番地	x1	10
8番地	x2	20
12番地	x3	30
:	:	:
400番地	x100	1000

10	4番地	10
20	8番地	20

## ポインタを使用したプログラム例2

```
#include <stdio.h>
int main(void)
{
    int a;
    int* b;    // ポインタ変数の宣言

    a=321;
    printf( "a=%d\n" ,a);

    b= &a;
    *b=123;
    printf("a=%d\n", a);
    return 0;
}
```

- &a : aが入っているアドレスを取り出す (& : アンド→「アドレス」と覚える)
- int \*b; ポインタ変数の宣言
- \*b : ポインタbが指す先(アドレス)の値 (\* : アスタリスク→「値」と覚える)

## ポインタを使用したプログラム例3

```
#include <stdio.h>
int main(void)
{
    int a, x;                // 通常変数の宣言
    int *pt1, *pt2;          // ポインタ変数の宣言
    a = 321;
    printf( "a=%d \n" ,a);
    pt1 = &a                // aのアドレスをpt1に代入
    *pt1 = 123;              // pt1が指すアドレスの内容に123を代入
    x=*pt1;                  // pt1が指すアドレスの内容をxに代入
    pt2= pt1;                // ポインタpt1をpt2に代入
    printf("a=%d, x=%d \n", a, x);
    printf("pt1 =%p, pt2=%p \n", pt1, pt2);
    return 0;
}
```

# ポインタの用途

関数の引数をポインタにする

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

変数aを格納する領域が  
メモリ上にできる

変数aのアドレスに入力  
値を入れる

キーボードから入力した値を変数aに代入する

関数scanf ( )に変数aのアドレスを教えるために引数をポインタにしている

# 関数の引数にポインタを使用する例

- 2つの変数(引数)の内容を交換する関数

## 誤ったプログラム

```
int main()
{
    int a=123, b=321;
    swap(a,b);
    ...
}

void swap(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

これらのa,bは互いに無関係

## 正しいプログラム

```
int main()
{
    int a=123,b=321;
    swap(&a,&b);
    ...
}

void swap(int *a , int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

これらのa,bはアドレスを通して結びついている

# 関数の引数にポインタを使用する例

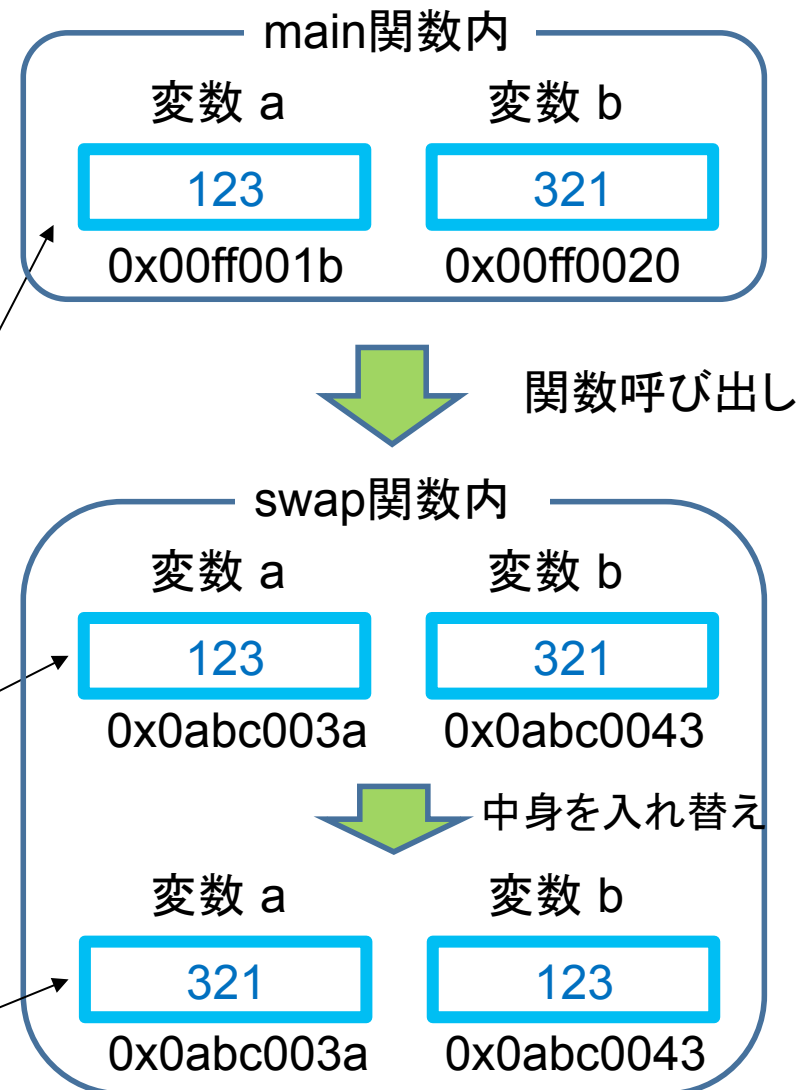
誤ったプログラム

```
int main()
{
    int a=123, b=321;
    swap(a,b);
    ...
}

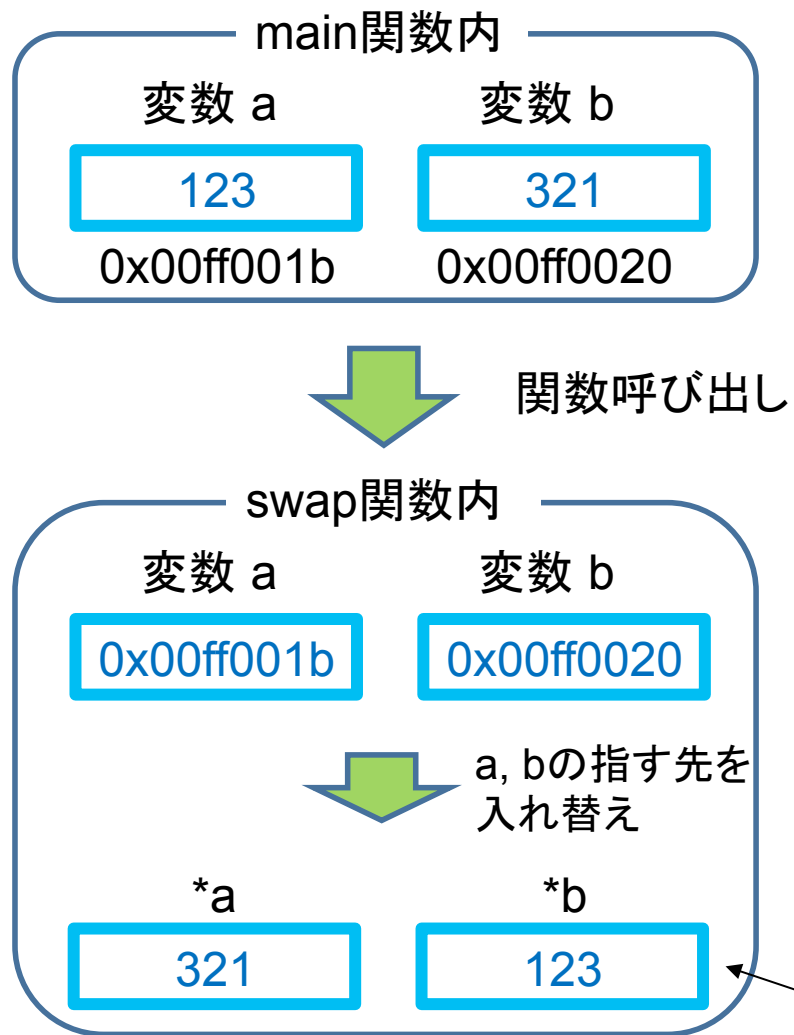
void swap(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

これらのa,bは互いに無関係  
格納されている場所(アドレス)が違う

main関数内の変数a,bは変更されない



# 関数の引数にポインタを使用する例



## 正しいプログラム

```
int main()
{
    int a=123,b=321;
    swap(&a,&b);
    ...
}
```

```
void swap(int *a , int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

これらのa,b  
はアドレスを  
通して結びつ  
いている

main関数内の変数a,bはポインタを  
使って書き換えられる



# ポインタと配列

- 配列の名前そのものは、配列の最初の要素を指すポインタ

```
int a[4];
```

← aはa[0]へのポインタを表す

- 2番目以降の要素を呼び出すには、ポインタを加算する

```
int* p=a+2;
```

← pはaから2つ先の要素a[2]を指す

- ポインタを使った配列の参照：

```
int b, c, d;  
b=*a;  
c=*(a+1);  
d=*(a+2);
```

– b=44, c=38, d=12と同じ

a[0]	44	← a
a[1]	38	← a+1
a[2]	12	← a+2
a[3]	25	← a+3

# 配列の初期化

## ポインタを使わない初期化

```
int main()
{
    int a[4];
    int i;
    for(i=0;i<4;i++){
        a[i]=i;
    }
    for(i=0;i<4;i++){
        printf("a[%d]=%d\n", i,a[i]);
    }
}
```

## ポインタを使った初期化

```
int main()
{
    int a[4];
    int i;
    for(i=0;i<4;i++){
        *(a+i)=i;
    }
    for(i=0;i<4;i++){
        printf("*(a+%d)=%d\n", i,*(a+i));
    }
}
```

# 配列とポインタの関係

```
int main(void)
{
    int a[20];
    // intの変数のサイズは4byte
    printf("Address is %p" \n ,a);
    return 0;
}
```

↑  
&a[0]と同じ

- 配列を宣言し、配列の先頭アドレスを表示するプログラム
- 配列宣言により、aはポインタ変数として使用可能
- 20個のintのかたまりと先頭アドレスが確保される
- ポインタが1つ進むと、intのサイズだけアドレスが進む

アドレス: 実際のメモリ上の位置

ポインタ: アドレスを操作しやすいように作った変数

このようにも書ける！



配列	ポインタ	アドレス
a[0]	*a	A(aのアドレス)
a[1]	*(a+1)	A+4
a[2]	*(a+2)	A+8
a[3]	*(a+3)	A+12
:	:	:
a[18]	*(a+18)	A+72
a[19]	*(a+19)	A+76

ほとんど同一のものとして扱うことが可能

int a[20]でなく、char a[20]とした場合は？

# ポインタと配列

```
int a[4];
```

← aはa[0]へのポインタを表す

変数名	値	アドレス	ポインタ
a[0]	44	0x11ffab02	← a
a[1]	38	0x11ffab06	← a+1
a[2]	12	0x11ffaba0	← a+2
a[3]	25	0x11ffaba4	← a+3

int型の場合,ポインタが1加算されると  
アドレスは4ずつ加算される