cp-11. ポインタ

（C プログラミング入門）

URL: https://www.kkaneko.jp/pro/adp/index.html

金子邦彦

内容

例題１．変数のメモリアドレス表示

例題２．配列のメモリアドレス

例題３．２次元配列のメモリアドレス

メモリとメモリアドレス

例題４．棒グラフを表示する関数

関数への配列の受け渡し

例題５．２次元配列の受け渡し

関数への配列の受け渡し

例題６．局所変数と仮引数のメモリアドレス

例題７．関数へのポインタ渡し

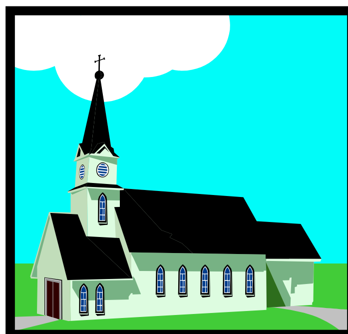
関数へのポインタ渡しとポインタ変数

目標

* データは，アドレス付けされて，メモリに入っていることを理解する
* ポインタ変数を使い，関数との情報の受け渡しができるようになる

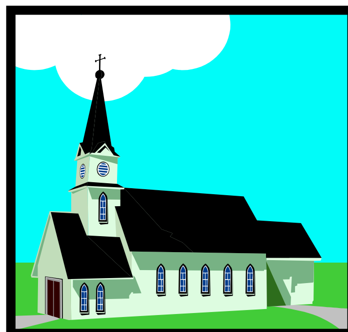
家と住所

Ａさんの家



福岡市東区
箱崎１丁目１番

Ｂさんの家



福岡市東区
箱崎２丁目２番

名前　　家　　住所

メモリアドレスとは

メモリアドレスとは

- すべてのデータには「メモリアドレス」が付けられている

変数の中身：　値

「１８」「１０７．７５」など

変数名：　　プログラム内で使うための名前

「age」，「rate」 など

メモリアドレス：　変数のそれぞれに付けられた「住所」の

ようなもの

例題１．変数のメモリアドレス表示

- 次の３つの変数を使って，「底辺と高さを読み込んで，面積を計算するプログラム」を作る.

底辺	teihen	浮動小数データ
高さ	takasa	浮動小数データ
面積	menseki	浮動小数データ

- これら変数のメモリアドレスの表示も行う

```
#include <stdio.h>
#pragma warning(disable:4996)
int main()
{
    double teihen,takasa,menseki;
    printf("teihen=");
    scanf("%lf", &teihen);
    printf("takasa=");
    scanf("%lf", &takasa);
    menseki = teihen * takasa * 0.5;
    printf("menseki = %f¥n",menseki);
    printf("address(teihen)  = %p¥n", &teihen );
    printf("address(takasa)  = %p¥n", &takasa );
    printf("address(menseki) = %p¥n", &menseki );
    return 0;
}
```

「&」はメモリアドレスの取得

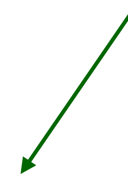
「%p」はメモリアドレスの表示

変数のメモリアドレス表示

実行結果の例

```
teihen=3
takasa=4
menseki = 6.000000
address(teihen) = 0065FDF0
address(takasa) = 0065FDE8
address(menseki) = 0065FDE0
```

表示された
メモリアドレス



メモリアドレス

メモリ

変数名	メモリ	メモリアドレス
menseki →	6.000000	0065FDE0
takasa →	4.000000	0065FDE8
teihen →	3.000000	0065FDF0

メモリアドレスの取得と表示

```
printf("address(teihen)  = %p¥n", &teihen );
```

%p → メモリアドレスの表示

&teihen → メモリアドレスの取得

- 変数からメモリアドレスの取得

  &: メモリアドレスを取得するための演算子

  変数名（など）の前に付ける

- メモリアドレスの表示のための書式

  %p: メモリアドレスを表示せよという指示

  printf 文などで使用

例題２．配列のメモリアドレス

- 次の２つの配列を使って，ベクトル（1.9, 2.8, 3.7）と，ベクトル（4.6, 5.5, 6.4）の内積を求めるプログラムを作る．

  ベクトル（1.9, 2.8, 3.7）　u　要素数３の浮動小数の配列
  ベクトル（4.6, 5.5, 6.4）　v　要素数３の浮動小数の配列

- これら配列の要素について，メモリアドレスの表示も行う

```
#include <stdio.h>
#pragma warning(disable:4996)
int main()
{
    double u[]={1.9, 2.8, 3.7};
    double v[]={4.6, 5.5, 6.4};
    int i;
    double ip;
    ip = 0;
    for (i=0; i<3; i++) {
        ip = ip + u[i]*v[i];
    }
    printf("内積=%f¥n", ip);
    printf("address(u[0]) = %p¥n", &u[0]);
    printf("address(u[1]) = %p¥n", &u[1]);
    printf("address(u[2]) = %p¥n", &u[2]);
    printf("address(v[0]) = %p¥n", &v[0]);
    printf("address(v[1]) = %p¥n", &v[1]);
    printf("address(v[2]) = %p¥n", &v[2]);
    return 0;
}
```

「&」はメモリアドレスの取得

「%p」はメモリアドレスの表示

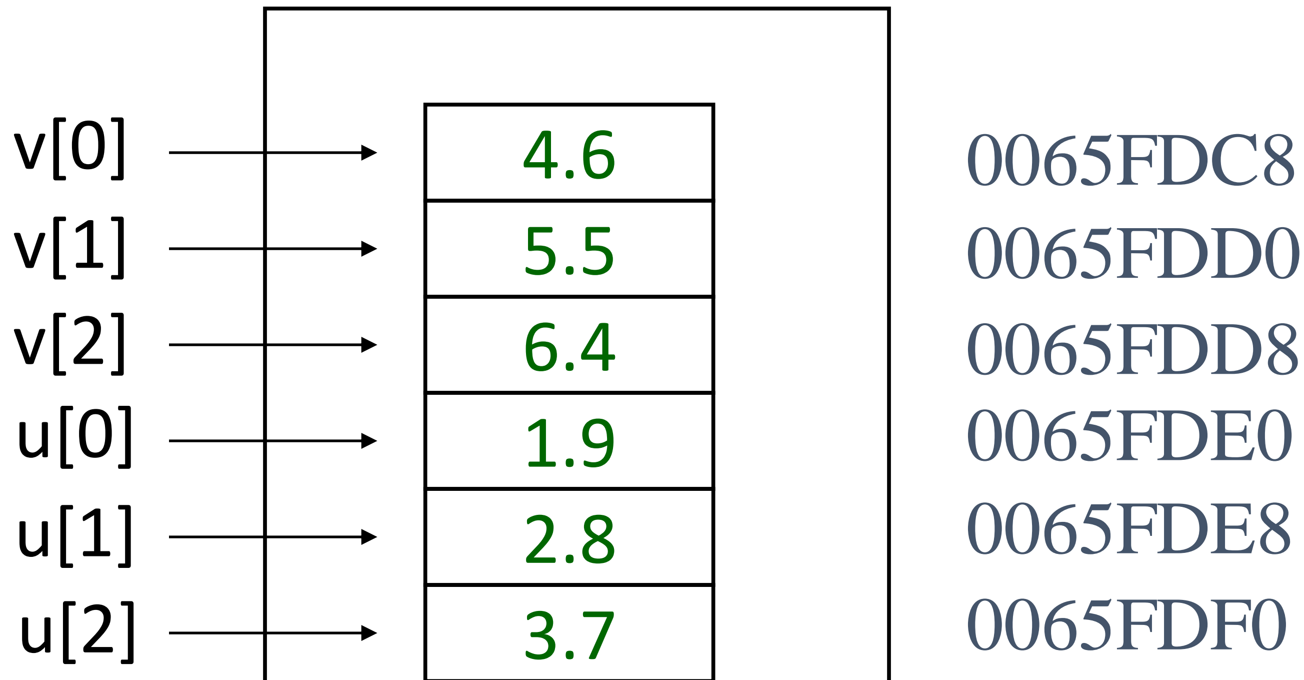
配列のメモリアドレス

実行結果の例

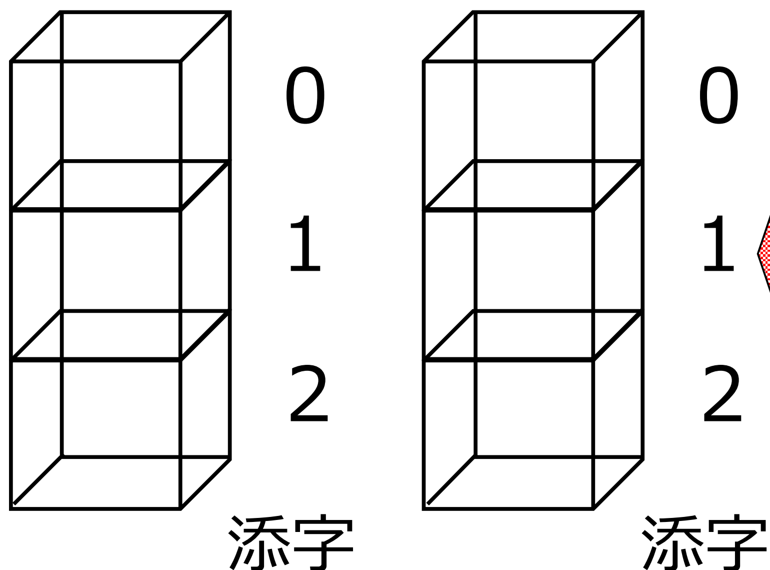
表示された
メモリアドレス

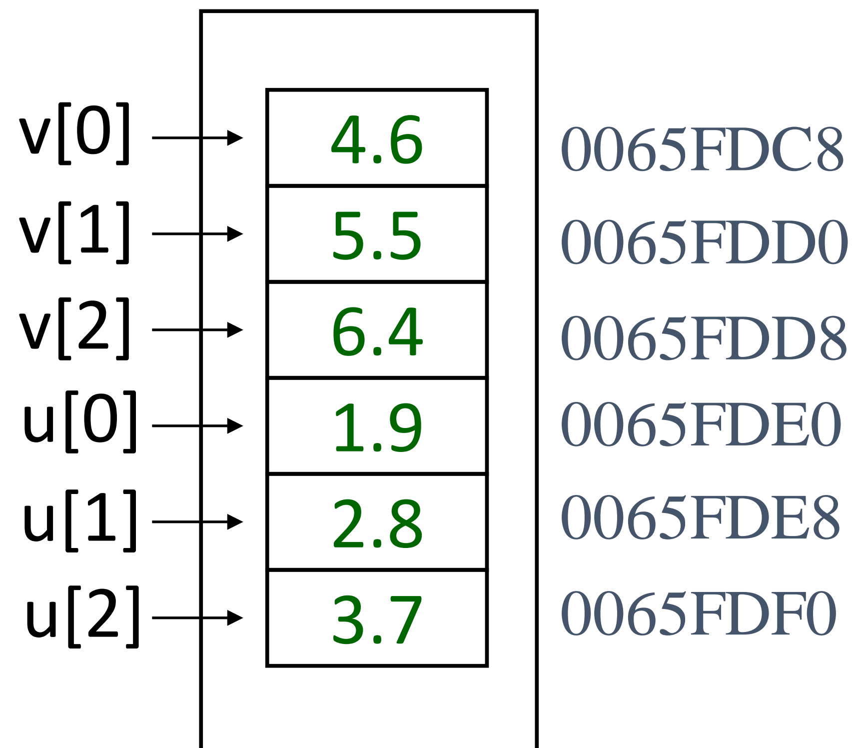
```
内積=47.820000
address(u[0]) = 0065FDE0
address(u[1]) = 0065FDE8
address(u[2]) = 0065FDF0
address(v[0]) = 0065FDC8
address(v[1]) = 0065FDD0
address(v[2]) = 0065FDD8
```

メモリアドレス



配列とメモリアドレス





２つの配列

メモリ内の配置
（配列の並びはそのままでメモリに入る）

例題３． ２次元配列のメモリアドレス

- 次の２つの配列を使って，２行３列の行列の和を求めるようなプログラムを作る.

  a　２行３列の行列　整数
  b　２行３列の行列　整数

- 配列ａの要素について，メモリアドレスの表示も行う

```
#include <stdio.h>
#pragma warning(disable:4996)
int main()
{
    int a[2][3]={{1,2,3},{4,5,6}};
    int b[2][3]={{9,8,7},{6,5,4}};
    int i;
    int j;
    for (i=0; i<2; i++) {
        for (j=0; j<3; j++) {
            printf("%d, ", a[i][j]+b[i][j]);
        }
        printf("¥n");
    }
    printf("address(a[0][0]) = %p¥n", &a[0][0]);
    printf("address(a[0][1]) = %p¥n", &a[0][1]);
    printf("address(a[0][2]) = %p¥n", &a[0][2]);
    printf("address(a[1][0]) = %p¥n", &a[1][0]);
    printf("address(a[1][1]) = %p¥n", &a[1][1]);
    printf("address(a[1][2]) = %p¥n", &a[1][2]);
    return 0;
}
```

「&」はメモリアドレスの取得

「%p」はメモリアドレスの表示

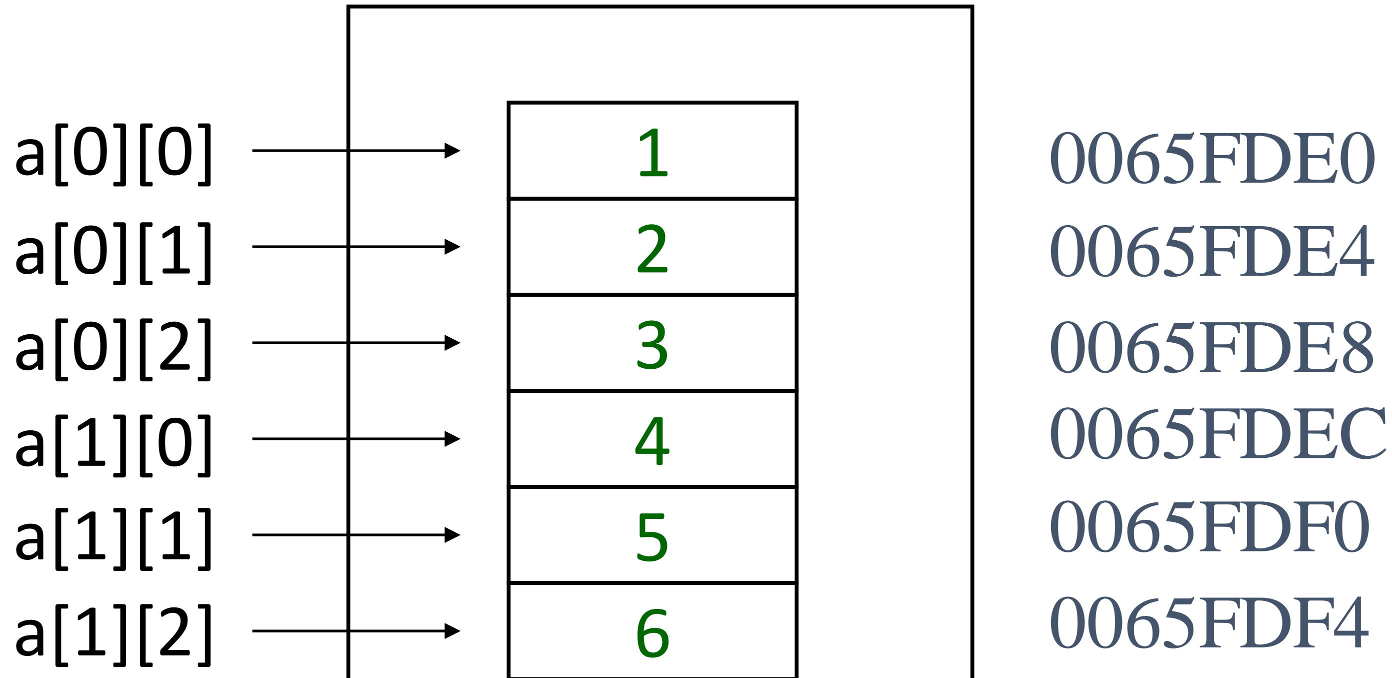
２次元配列のメモリアドレス

実行結果の例

表示された
メモリアドレス

```
10, 10, 10,
10, 10, 10,
address(a[0][0]) = 0065FDE0
address(a[0][1]) = 0065FDE4
address(a[0][2]) = 0065FDE8
address(a[1][0]) = 0065FDEC
address(a[1][1]) = 0065FDF0
address(a[1][2]) = 0065FDF4
```

メモリアドレス

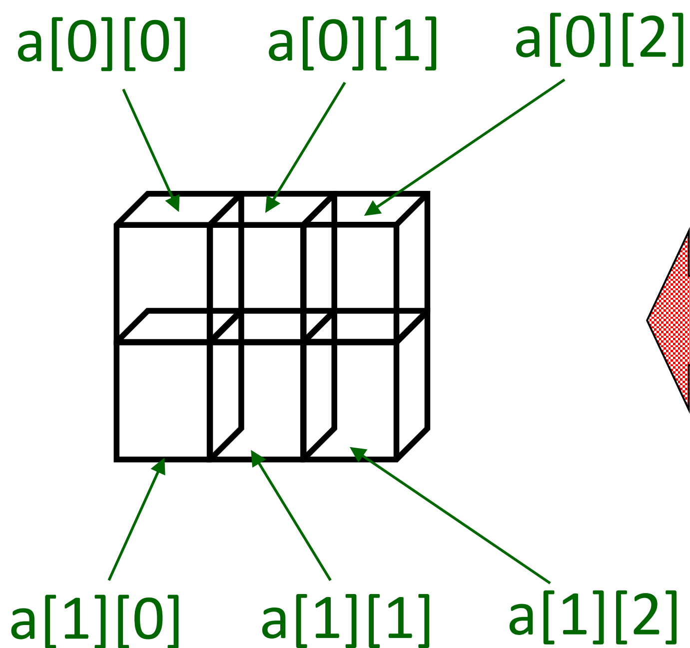


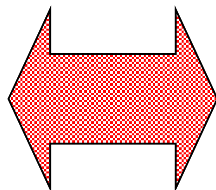
0065FDE0
0065FDE4
0065FDE8
0065FDEC
0065FDF0
0065FDF4

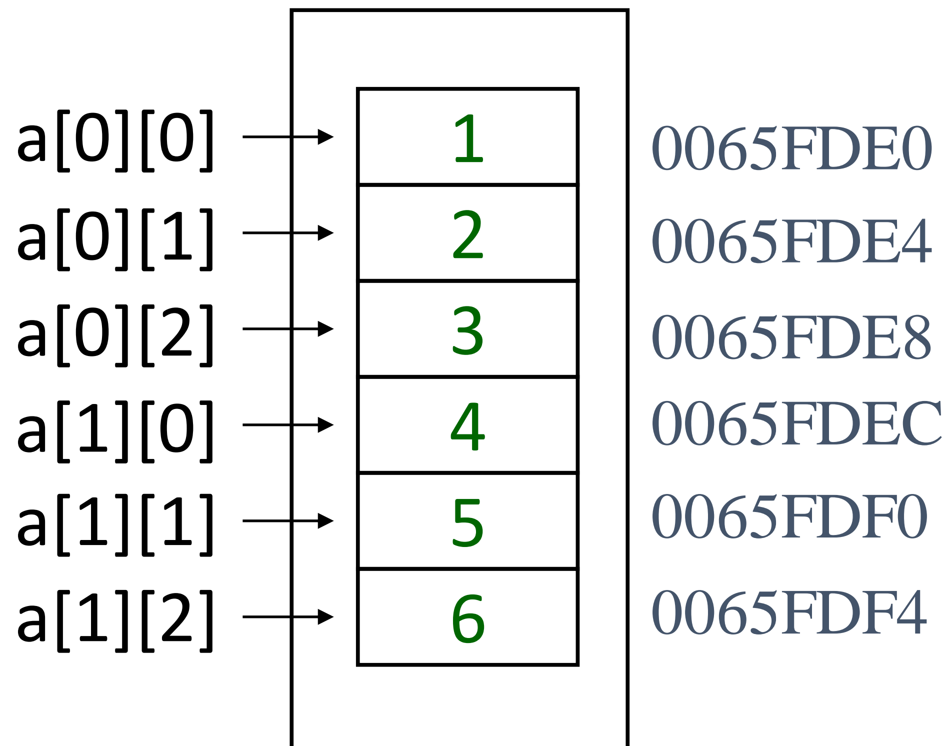
メモリアドレス

２次元配列とメモリアドレス

２次元配列 a







メモリ内の配置

（a[0][0] → a[0][1] → a[0][2] → a[1][0] → a[1][1] → a[1][2] の順で入る）

２次元配列

例題４． 棒グラフを表示する関数

- 整数の配列から，その棒グラフを表示する bar_graph 関数を作る．同時に， bar_graph 関数を呼び出すmain関数も作る
  - 整数の配列及び配列のサイズをbar_graph関数に渡すこと
  - bar_graph関数の返り値はない（void とする）

```
#include <stdio.h>
#pragma warning(disable:4996)
void bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    return;
}
void bar_graph( int len, int x[] )
{
    int i;
    for (i=0; i<len; i++) {
        bar( x[i] );
    }
    return;
}
int main()
{
    int a[7]={6,4,7,1,5,3,2};
    bar_graph( 7, a );
    return 0;
}
```

「整数の配列の先頭要素のメモリアドレス」を受け取って，配列 x として使う．

配列の先頭要素のメモリアドレスはすでに受け取ったので，x[i] のように書いて配列の要素を使える

配列 a の先頭要素のメモリアドレス（&a[0] の省略形）

棒グラフを表示する関数

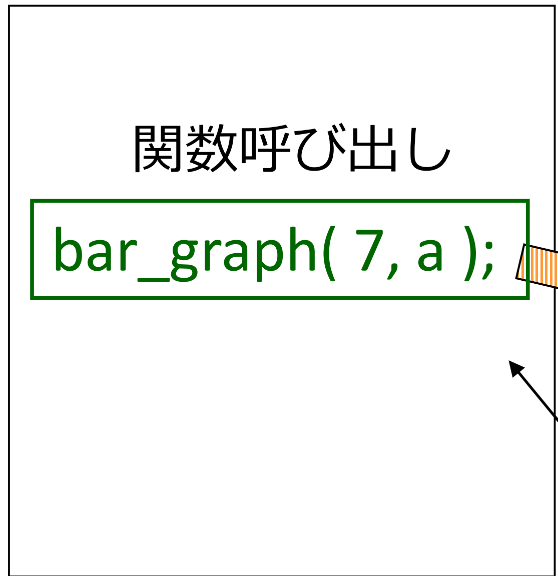
実行結果の例

```
*****
****
*******
*
*****
***
**
```

関数呼び出しの流れ

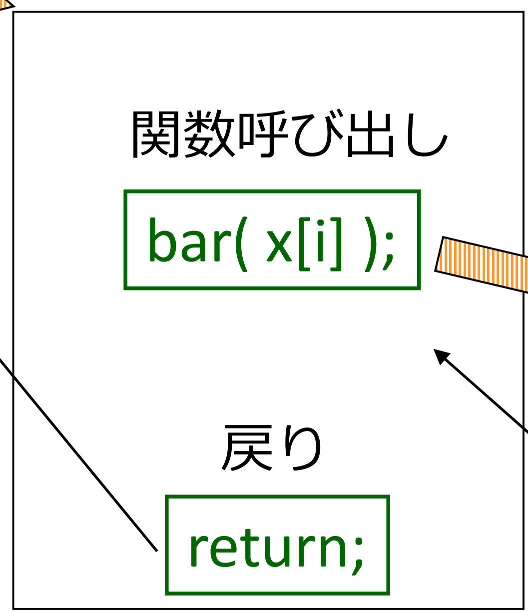
main 関数

int main()



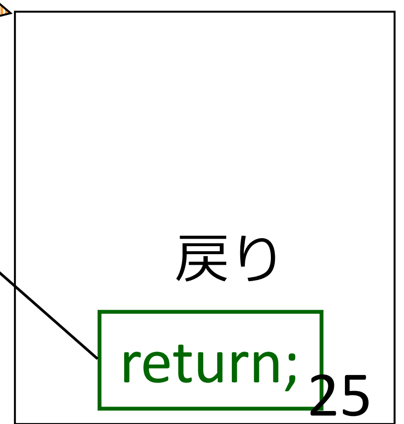
bar_graph 関数

void bar_graph(int len, int x[])



bar 関数

void bar(int len)



関数への配列の受け渡し

- 呼び出し側
  - 配列変数名を書いて，配列の先頭メモリアドレスを，関数に渡す

例）　bar_graph(7, a);

配列 a の先頭要素のメモリアドレス
（&a[0] の省略形）

- 関数側
  - 配列を受け取る（正確には，配列の先頭メモリアドレス）ことを宣言しておく

void bar_graph(int len, int x[])

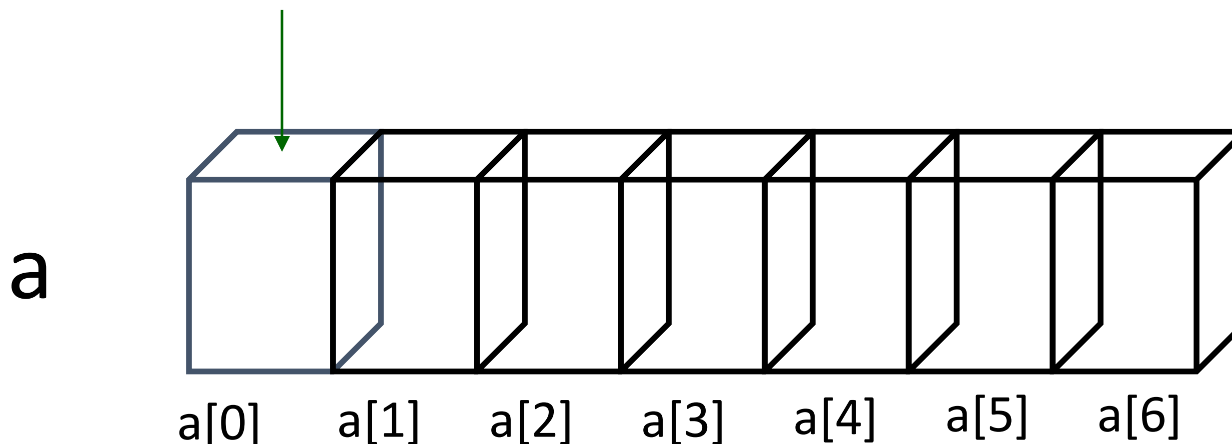
「整数の配列の先頭要素の
メモリアドレス」を受け取って，
配列 x として使う．

  - 受け取った配列は，普通と同じに使える

配列とポインタ

プログラム例：bar_graph(7, a);

配列の先頭要素



- プログラム中に配列名を単独（例えば「a」）で書くと，配列の先頭要素のメモリアドレスという意味

課題１．ベクトルの内積

- ２つの３次元ベクトルの内積を求める関数 product を作成しなさい．同時に，product 関数を使う main 関数を作成し，正しく動作することを確認すること.
    - ２つの３次元ベクトルをproduct関数に渡すこと
    - product関数の返り値として，求めた内積を返すこと
    - 第５回の講義資料の「ベクトルの内積」の部分を参考にして下さい

例題５．　２次元配列の受け渡し

- ２次元配列の先頭要素のメモリアドレスと，配列の大きさから，配列の中身を表示する関数を作る.

2次元配列の受け渡し

```
#include <stdio.h>
#pragma warning(disable:4996)
void print_matrix( int *x, int n, int m ) {
   int i;
   int j;
   for( i = 0; i < n; i++ ) {
      for( j = 0; j < m; j++ ) {
         printf( "%d, ", x[i*m+j] );
      }
      printf( "¥n" );
   }
   return;
}
int main()
{
   int a[2][2] = {{1,2},{3,4}};
   print_matrix( a, 2, 2 );
   return 0;
}
```

「整数データのメモリアドレス」を受け取って，x として使う.

2 次元配列 x の i 行 j 列目

配列 a の先頭要素のメモリアドレス

関数呼び出しの流れ

main 関数

int main()

関数呼び出し

```
print_matrix( a, 2, 2 );
```

print_matrix 関数

```
void print_matrix( int *x, int n, int m);
```

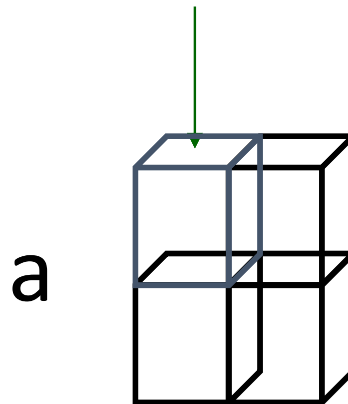
戻り

```
return;
```

２次元配列とポインタ

プログラム例： print_matrix(a, 2);

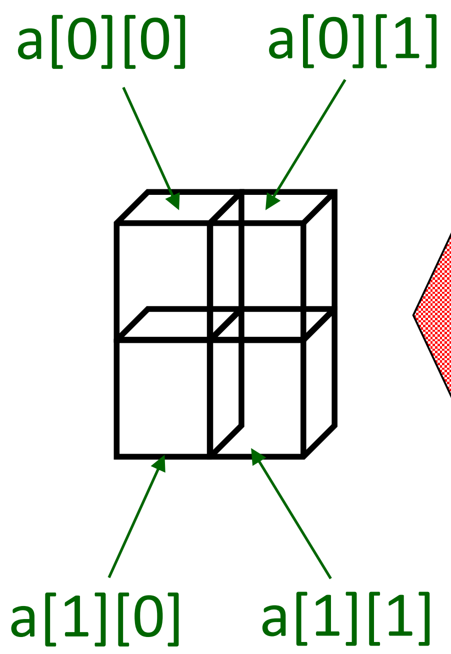
２次元配列の先頭要素（つまり a[0][0]）



- ２次元配列の場合でも，プログラム中に配列名を単独で書くと，配列の先頭要素のメモリアドレスという意味

x[i*n+j] の意味

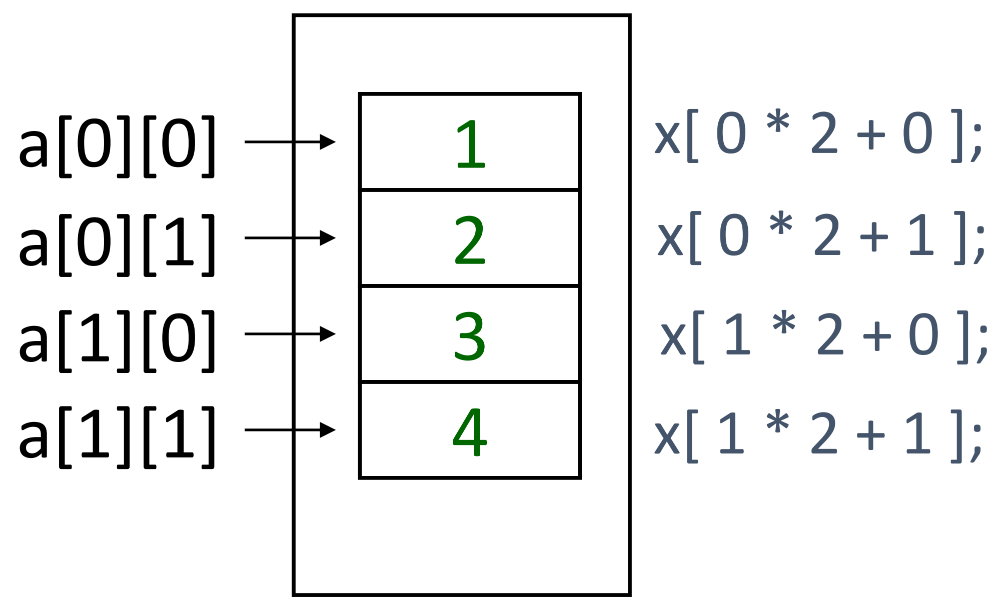
２次元配列 a



２次元配列

配列の名前 a で
使用する場合の書き方
（main 関数内）

配列の先頭アドレスが，
ポインタ変数 x に
入っている場合の書き方



メモリ

課題２． ２つの行列の和

- ２つの行列の和を求める関数 add_matrix を作成しなさい．同時に， add_matrix 関数を使う main 関数を作成し，正しく動作することを確認すること.
  - add_matrix関数に渡されるのは次の通り

1. 和を求めるべき２つの行列
2. 行列の縦，横の大きさ
3. 求めた和を格納すべき行列

例題６．局所変数と仮引数のメモリアドレス

- 整数から，その長さだけの棒を表示するbar 関数と，bar関数を呼び出すmain関数を作る
- 局所変数と仮引数について，メモリアドレスを表示することも行う

```
#include <stdio.h>
#pragma warning(disable:4996)
void bar( int len )
{
    int i;
    for (i=0; i<len; i++) {
        printf("*");
    }
    printf("¥n");
    printf("address(len) = %p¥n", &len);
    printf("address(i) = %p¥n", &i);
    return;
}

int main()
{
    int len;
    printf( "len=" );
    scanf( "%d", &len );
    bar( len );
    printf("address(len) = %p¥n", &len);
    return 0;
}
```

仮引数（パラメータ）

局所変数

局所変数

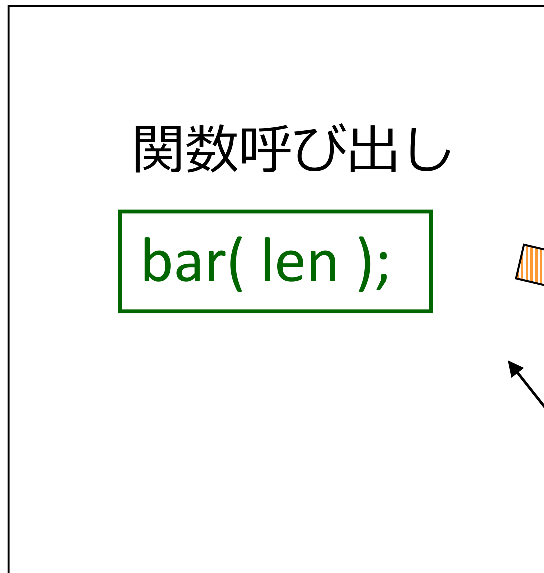
局所変数と仮引数のメモリアドレス

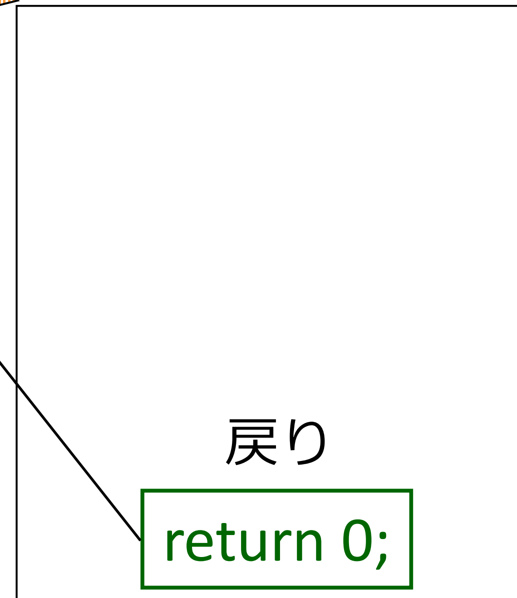
実行結果の例

```
len=10
**********
address(len) = 0065FDA4
address(i) = 0065FD98
address(len) = 0065FDF4
```

表示されたメモリアドレス

関数呼び出しの流れ





メモリアドレス

メモリ

	値	メモリアドレス
i	0	0065FD98
len	10	0065FDA4

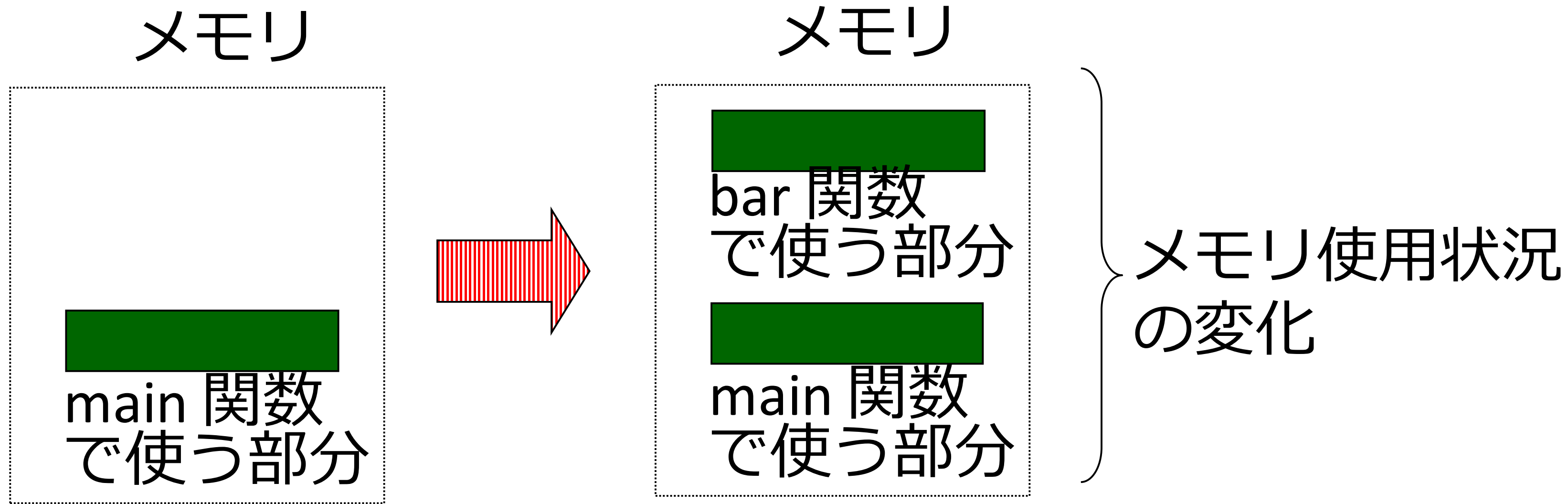
bar 関数で使う部分

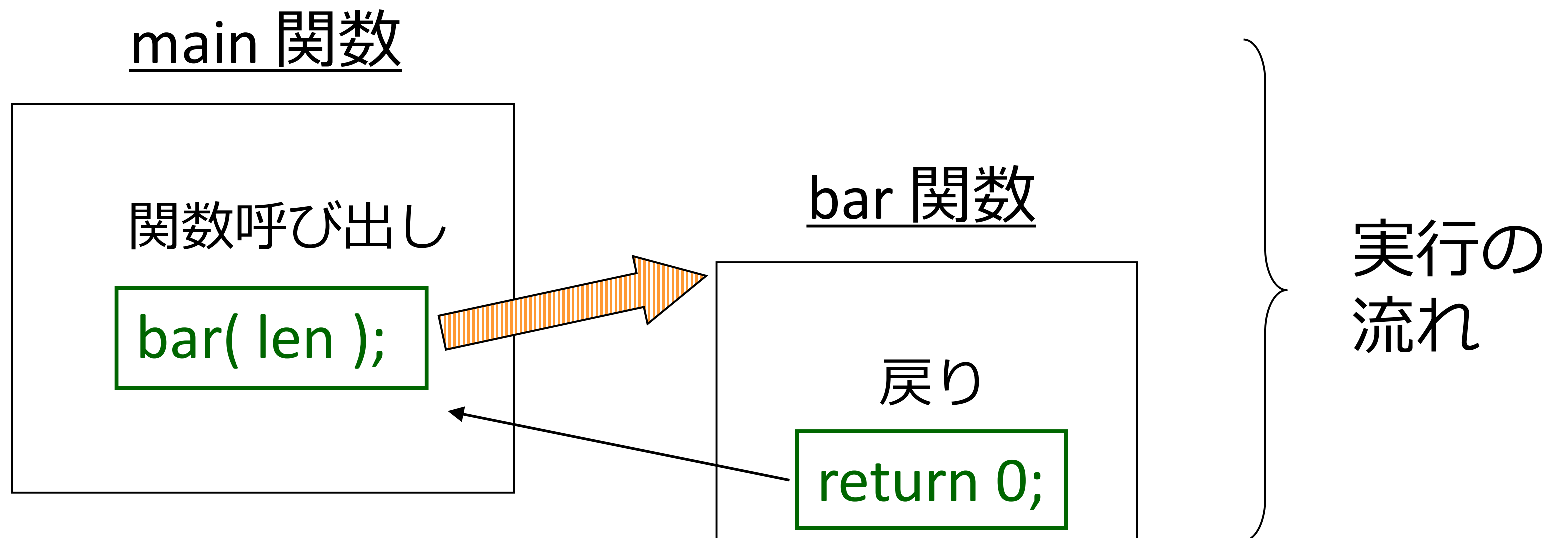
	値	メモリアドレス
len	10	0065FDF4

main 関数で使う部分

メモリアドレス

関数呼び出しに伴うメモリ使用状況の変化





例題７．関数へのポインタ渡し

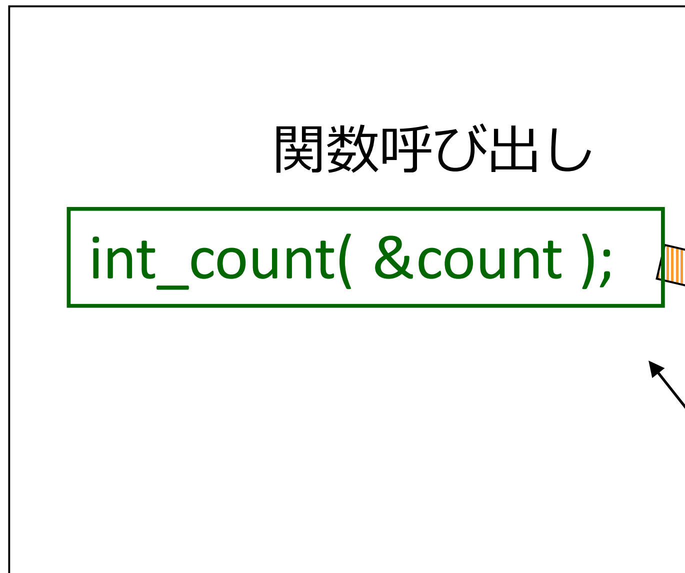
- 呼び出し側の局所変数を書き換えてしまうような関数を作る

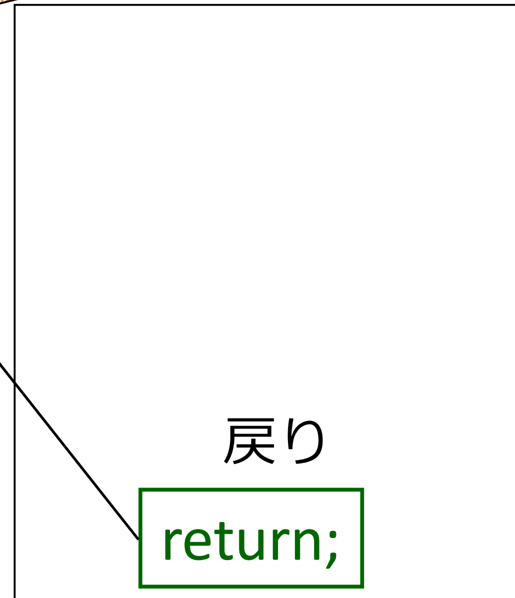
```
#include <stdio.h>
#pragma warning(disable:4996)
void int_count(int *count_ptr)
{
    *count_ptr = *count_ptr + 1;
    return;
}
int main()
{
    int count = 0;
    while ( count < 10 ) {
        int_count(&count);
    }
    printf( "count=%d¥n", count );
    return 0;
}
```

仮引数: (int *count_ptr)

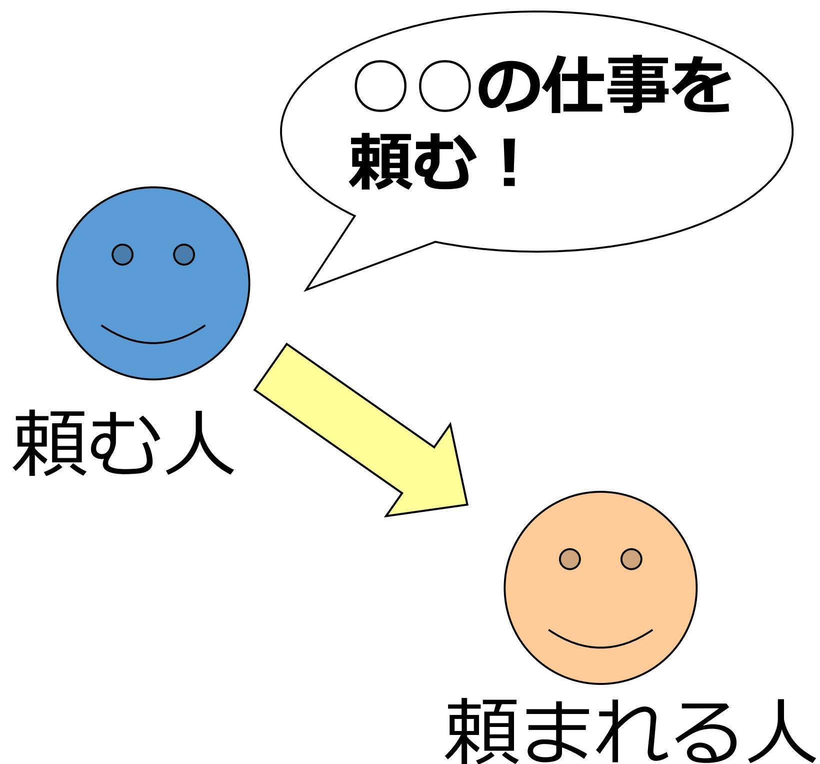
局所変数: int count = 0;

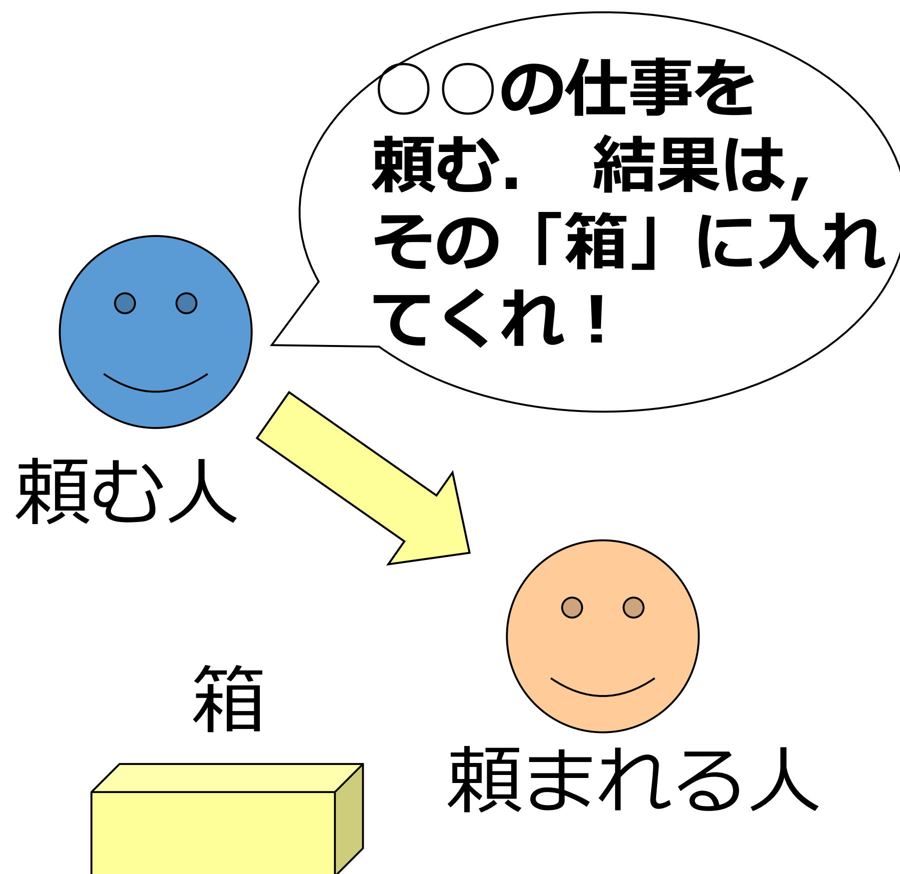
関数呼び出しの流れ





仕事の依頼



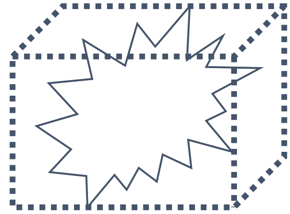


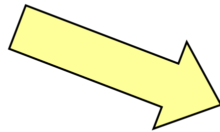
- 一方通行の場合
- 返事を受け取りたい場合

関数へのポインタ渡し

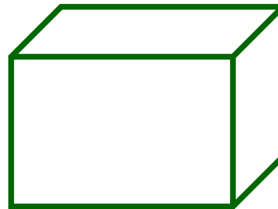
```
int_count(&count)
```

count へのポインタ
(&count で得られる)





変数 count



呼び出し側

- 関数 int_count の呼び出しで，&count （変数 count へのポインタ）を渡す

関数にローカルなポインタ変数

int_count(int *count_ptr)

count へのポインタ

中身がコピーされる

count_ptr

count_ptr という名前の付いたポインタ変数（関数の中でのみ使用）

変数 count

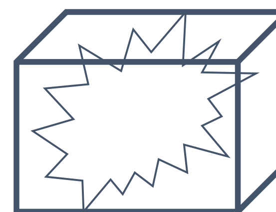
呼び出し側

呼び出され側

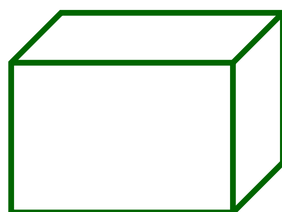
- 関数に渡された &count は，ポインタ変数 count_ptr に格納される

ポインタ変数によるデータ操作

```
*count_ptr = *count_ptr +1;
```



関数の中から見ると，
*count_ptr はこれ



呼び出し側

呼び出され側

- count_ptr が指している変数 count の値が 1 増える

メモリ

count_ptr → [ ] } メモリアドレスが入る変数

int_count 関数で使う部分

count → [ ] } 整数データが入る変数

main 関数で使う部分

関数へのデータの受け渡し

	ポインタ変数を使わない場合	ポインタ変数を使う場合
関数に受け渡されるもの	変数の「中身」 変数の値そのもの (call by valueという)	ポインタ変数の「中身」 ある変数への ポインタ (call by reference という)
性質	一方通行（渡された変数の書き換え不可能）	渡された変数の書き換え可能

ポインタ変数

age　１８　←　age_ptr

rate　107.75　←　rate_ptr

変数の名前　変数　ポインタ変数　ポインタ変数の名前

ポインタ変数

- 変数：　数や文字を格納

　（例）　int age;

　　　　double rate;

- ポインタ変数：ポインタを格納

　（例）　int *count_ptr

- ポインタ変数も名前を持つ

　（普通の変数と同じ）

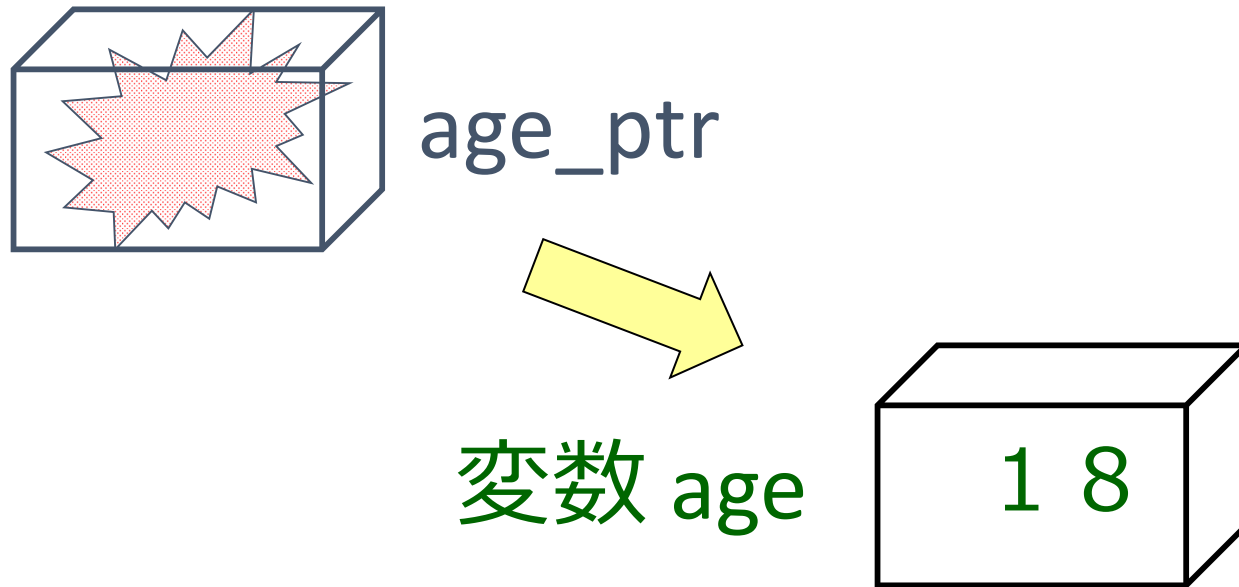
ポインタ変数の宣言

- 変数名の前に　＊　を付ける

例）　　int *age_ptr

ポインタ変数＝＆変数

プログラム例：　age_ptr = &age;



- ポインタ変数 age_ptr に，変数 age へのポインタをセットする

変数＝＊ポインタ変数

プログラム例：　x ＊ age_ptr;

- 変数 x に，ポインタ変数 age が指している変数の値をセットする

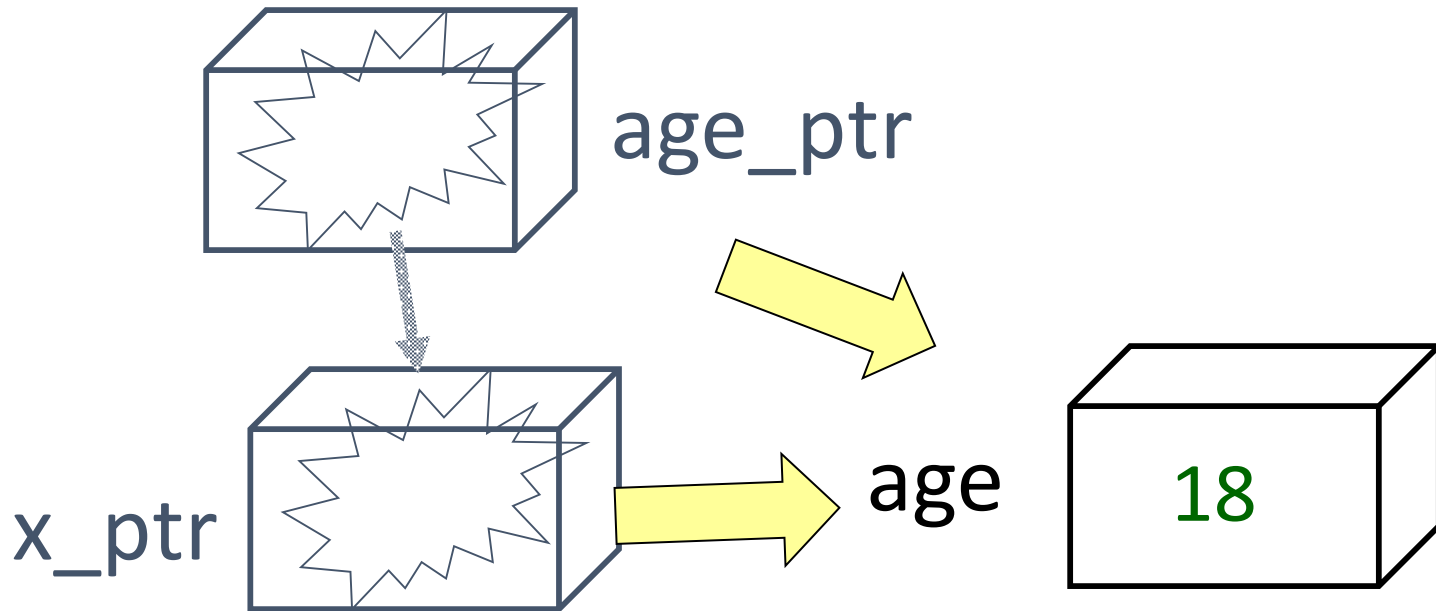
＊ポインタ変数＝「値」

プログラム例： *age_ptr = 22;

- ポインタ変数 age_ptr が指している変数 age に，値「２２」をセットする

ポインタ変数＝ポインタ変数

プログラム例： x_ptr = age_ptr;



- ポインタ変数 x_ptr に，age_ptr と同じポインタをセットする

ポインタを使ったプログラム例

```
#include <stdio.h>
#pragma warning(disable:4996)
int main()
{
    int age;
    int *age_ptr;

    age_ptr = &age;        } &を使用
    *age_ptr = 22;         } ＊を使用
    printf( "age = %d¥n", age );
    return 0;
}
```

- age_ptr は age を指しているから，age は「２２」に変わる

ポインタを使ったプログラム例

```
#include <stdio.h>
#pragma warning(disable:4996)
int main() {
    int age;
    int *age_ptr;
    int *second_ptr;

    age_ptr = &age;                 } &を使用
    second_ptr = age_ptr;           } ＊を使用
    *second_ptr = 22;
    printf( "age = %d¥n", age );
    return 0;
}
```

- いくつかのポインタ変数が，同じものを指していてもかまわない

scanf に & を付ける理由

- scanf では，変数に & を付けることになっていた

```
scanf("%lf", &teihen);
```

書式　&　読み込むべき変数名

- scanf は，データを読み込んだら，「メモリアドレス」を使って，読み込んだデータをメモリに置く

課題３．スタック

- スタックの push 関数，pop 関数及び中身を表示する関数を作成しなさい．同時に，これら関数を使う main 関数を作成し，正しく動作することを確認すること．但し，大域変数は使わないこと
  - main 関数の中で，配列及びスタックポインタの宣言を行うこと
  - push 関数，pop 関数内では，スタックポインタの増減を正しく行うこと（ポインタ変数を使用すること）

ptr++ の意味

```
int ary[7];
int *ptr;
ptr = &a[0];
ptr++;
ptr++;
printf ( “%d”, *ptr );
```

ポインタ変数 ptr に，配列 ary へのポインタをセット

ptr を１つ動かす。（a の次の要素 a[1] を指す)

ptr を１つ動かす。（a の次の要素 a[2] を指す)

２次元配列での ptr++ の意味

```
int a[1000][1000];
int *ptr;
ptr = &a[99][0];
```

ポインタ変数 ptr に，配列 ary へのポインタをセット

```
ptr++;
```

ptr を１つ動かす。
（a の次の要素 a[99][1] を指す)

```
ptr++;
```

ptr を１つ動かす。
（a の次の要素 a[99][2] を指す)

```
printf ( "%d", *ptr );
```

＊を使って、値を取り出す

文字列とポインタ

プログラム例：　char string[6] = “March”;
char *ptr = &string[0];

ptr

文字列 string への
ポインタがセットされる

string

M	a	r	c	h	¥0

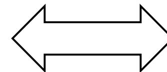
文字列の終わり
を示す記号

ポインタ変数 ptr に，文字列 string へのポインタを
セットする (char* ptr = string; と書いてもよい)

typedef

- typedef を使って，新しい型の名前を使えるようになる

```
struct date {
    int year;
    int month;
    int day
};
struct date a;
a.year = 2002;
a.month = 10;
a.day = 20;
```



同じ意味

```
typedef struct {
    int year;
    int month;
    int day
} date;
date a;
a.year = 2002;
a.month = 10;
a.day = 20;
```