ポインター

～プログラムを嫌いになるまで～

夜食

#注意事項#

* ポインターを理解するのは難しいです。
* 会長は理解していない模様です。

がんばりましょう。

手遅れになる前にお近くの先輩へ！

ポインターとは？

* メモリを指すもの。
* メモリの構成はアドレスとオブジェクトによって構成される。

アドレス	オブジェクト
0000	ア
0001	イ
0002	ス
0003	テ
0004	ィ
…	…
…	…
9998	あ
9999	あ
…	…

ポインターの性質1

* メモリを節約できる！
  →これは指す内容が大きくなればなるほど顕著

節約

ア	イ	ス	テ	ィ	ー	¥	o	ト	マ	ト	¥
o	猫	犬	虎	¥	o	て	つ	★	¥	o	ホ
モ	¥	o	ア	イ	ス	テ	ィ	ー	¥	o	ハ
ン	バ	ー	ガ	ー	¥	o	オ	ム	ラ	イ	ス
¥	o	ア	イ	ス	テ	ィ	ー	¥	o	回	鍋
肉	¥	o	ア	イ	ス	テ	ィ	ー	¥	o	一
平	ソ	バ	¥	o	ガ	ン	マ	¥	o	ラ	ー
メ	ン	¥	o	ア	イ	ス	テ	ィ	ー	¥	o
象	で	も	眠	る	睡	眠	薬	¥	o	ア	イ
ス	テ	ィ	ー	¥	o	あ	あ	あ	あ	あ	あ

節約

ア	イ	ス	テ	ィ	ー	¥	o	ト	マ	ト	¥
o	猫	犬	虎	¥	o	て	つ	★	¥	o	ホ
モ	¥	o	ア	イ	ス	テ	ィ	ー	¥	o	ハ
ン	バ	ー	ガ	ー	¥	o	オ	ム	ラ	イ	ス
¥	o	ア	イ	ス	テ	ィ	ー	¥	o	回	鍋
肉	¥	o	ア	イ	ス	テ	ィ	ー	¥	o	一
平	ソ	バ	¥	o	ガ	ン	マ	¥	o	ラ	ー
メ	ン	¥	o	ア	イ	ス	テ	ィ	ー	¥	o
象	で	も	眠	る	睡	眠	薬	¥	o	ア	イ
ス	テ	ィ	ー	¥	o	あ	あ	あ	あ	あ	あ

節約

ア	イ	ス	テ	ィ	ー	¥	o	ト	マ	ト	¥
o	猫	犬	虎	¥	o	て	つ	★	¥	o	ホ
モ	¥	o	o	o	¥	o	ハ	ン	バ	ー	ガ
ー	¥	o	オ	ム	ラ	イ	ス	¥	o	o	o
¥	o	回	鍋	肉	¥	o	o	o	¥	o	一
平	ソ	バ	¥	o	ガ	ン	マ	¥	o	ラ	ー
メ	ン	¥	o	o	o	¥	o	象	で	も	眠
る	睡	眠	薬	¥	o	o	o	¥	o	あ	あ
あ	あ	あ	あ								

ポインターの性質2

* 対象を動かさずに表現できる！
  →動かすわけには行かないデータなどへ対応

現実世界で考えてみる

* 家がほしい→家を見に行く
* 家がほしい→動きたくない→家の写真を見る
  この写真がポインターと言える

ポインターの性質3

* C では、引数付きの関数を呼び出す際、実引数（呼ぶ側の値）から仮引数（呼ばれ側の変数）へ値をコピーします。 つまり、引数の値のやりとりは、「呼び出し元→呼び出し先」の一方通行しかありません。
* 呼び出し先の関数から返してもらいたい値が1個だけなら return 文で返せますが、複数個を返したい場合、これでは困ってしまいます。
* 複数の値を関数間でやりとりしたい、しかもそれらを引数にしたい、という場合は、呼ばれ側の関数の仮引数をポインタ型にしておきます。
* 呼ぶ側では、やりとりしたい対象のアドレスを実引数として渡します。 そのアドレス値は、呼ばれ側の仮引数―ポインタ型で―にコピーされます。

ポインターの性質3

* ポインターを使うと……

関数で変更した値を元の変数でも反映させられる。

* これはプログラムでのポインターの書き方で説明

メモリ={アドレス,オブジェクト}

* 変数を作成したらアドレスが割り振られる

* 例) int number;
  number=5;

address	object
…	
1000	
1001 (number)	5
1002	
…	

C言語でのポインター

* ポインター型変数の定義
  型名*　変数名;
  例）int*　　int_pointer;
  　　int　　*int_pointer;
  (書き方が違うだけで内容は同じ)

* アドレス演算子 (&)
  例) scanf("%d",&value);
  →アドレスと対応させる。

アドレス	オブジェクト
…	
1000	
1001(value)	
1002	
…	

Valueのアドレスが1001のとき

scanf()でここに入る

scanf()　& アドレス演算子

* アドレス演算子 (&)

　例) scanf(“%d”,&value);

この時、scanfのときに3と入力したらアドレス1001の値はどうなるか…？

アドレス	オブジェクト
…	
1000	
1001(value)	3
1002	
…	

ポインター型演算子

ポインターの初期化

```
int number = 1111;
int       *int_pointer;
int_pointer=&number;
```

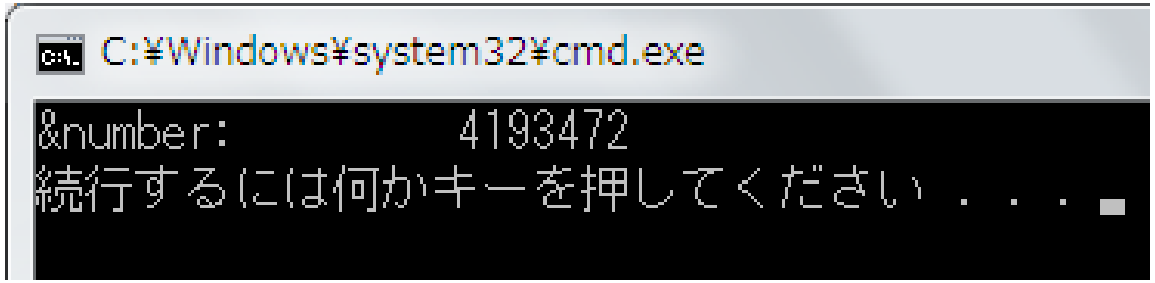
address	object
…	
???? (number)	1111
…	
1000	
1001 (int_pointer)	numberの アドレス
1002	
…	

ポインター型演算子

* じゃあnumberアドレスって何番よ？

printf(“number :%d”,&number);

これでnumberのアドレスが見れる



※変数は作成されたら

自動でアドレスを割り振られる

address	object
…	
4193472 (number)	1111
…	
1000	
1001 (int_pointer)	numberのアドレス
1002	
…	

ポインター型演算子

* ポインターに入れた値を使うぞ！

printf(“int_pointerの値は:%d¥n”,int_pointer);

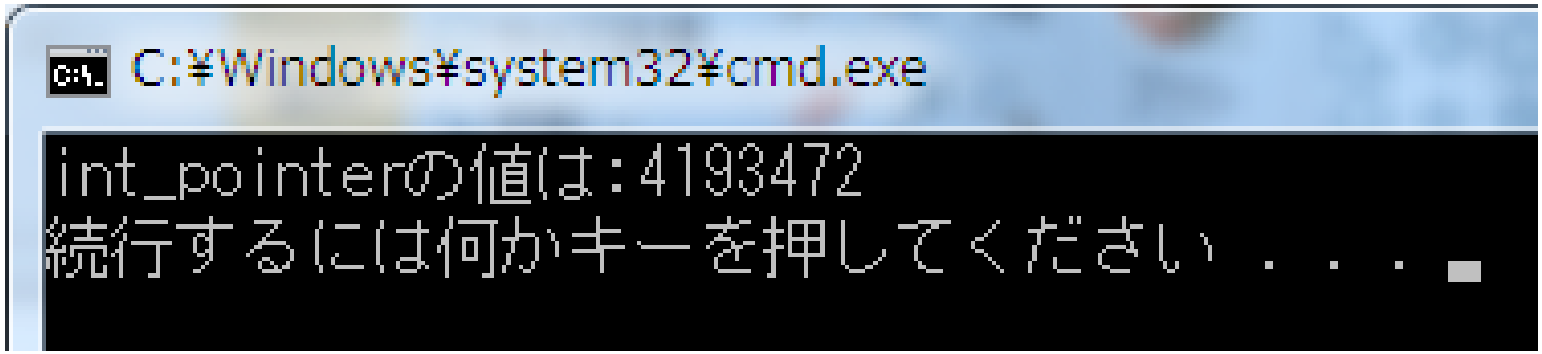
この時結果はどうなるか？

1. int_pointerの値は:1111
   (numberのオブジェクト)
2. int_pointerの値は:4193472
   (numberのアドレス)

address	object
…	
4193472 (number)	1111
…	
1000	
1001 (int_pointer)	(4193472)
1002	
…	

ポインター型演算子

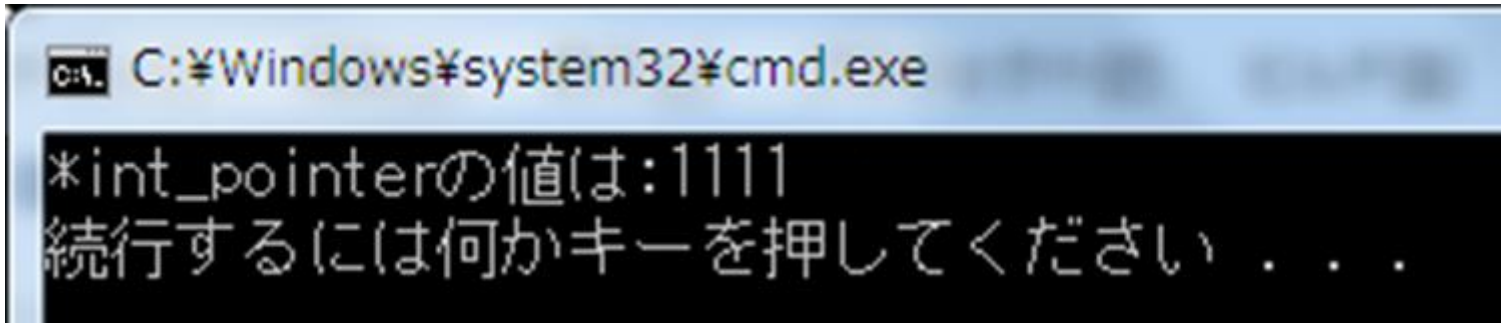
printf(“int_pointerの値は:%d¥n”,int_pointer);



答え:2

ポインター型演算子

* printf(“int_pointerの値は:%d¥n”,int_pointer);
* ではnumberの値を出すためには？
* printf(“*int_pointerの値は:%d¥n”,*int_pointer);



ポインター型演算子

* int_pointerはaddressを指していた。
* *int_pointerはobjectを指していた。

* このように、addressの中身を見たい場合 *int_pointer のようにポインター型変数の前に*を付ける。

address	object
…	
4193472 (number)	1111
…	
1000	
1001 (int_pointer)	(4193472)
1002	
…	

ポインター型演算子

注意事項

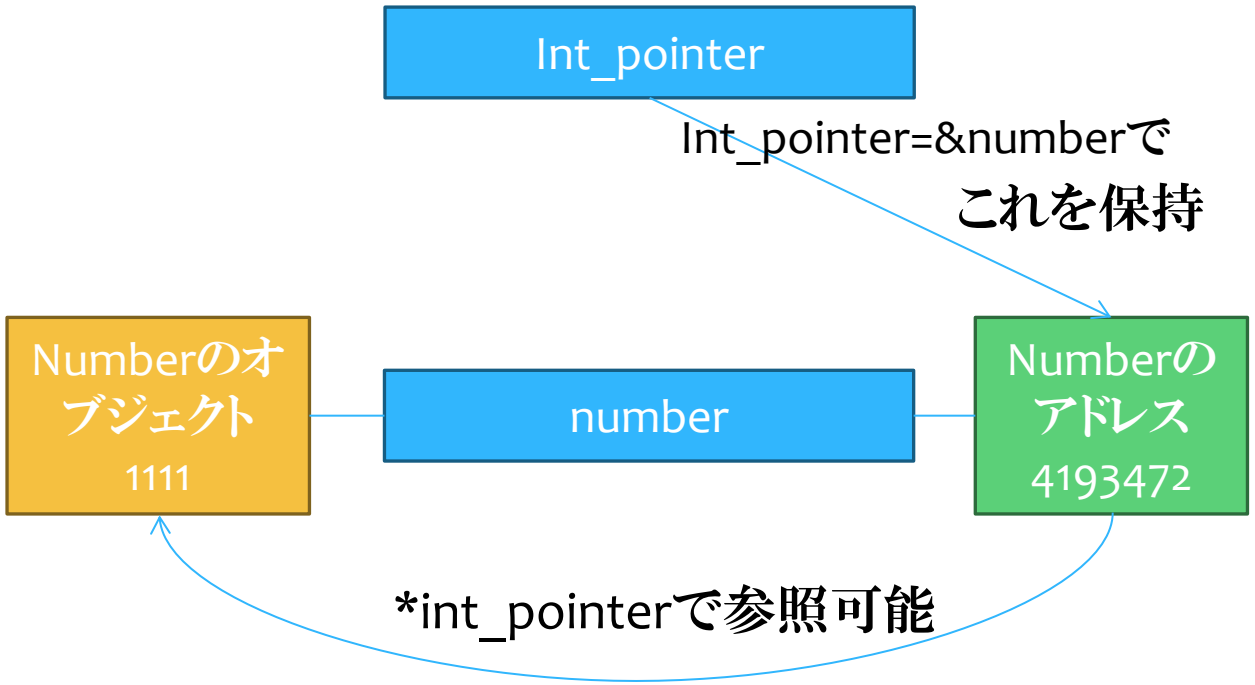
変数定義の　int　*int_pointer
中身参照の　*int_pointer

* これらはまったくの別物と考えてください。
  スイカって言われても”西瓜”と”Suica”があるみたいなもんです。

ポインター型演算子

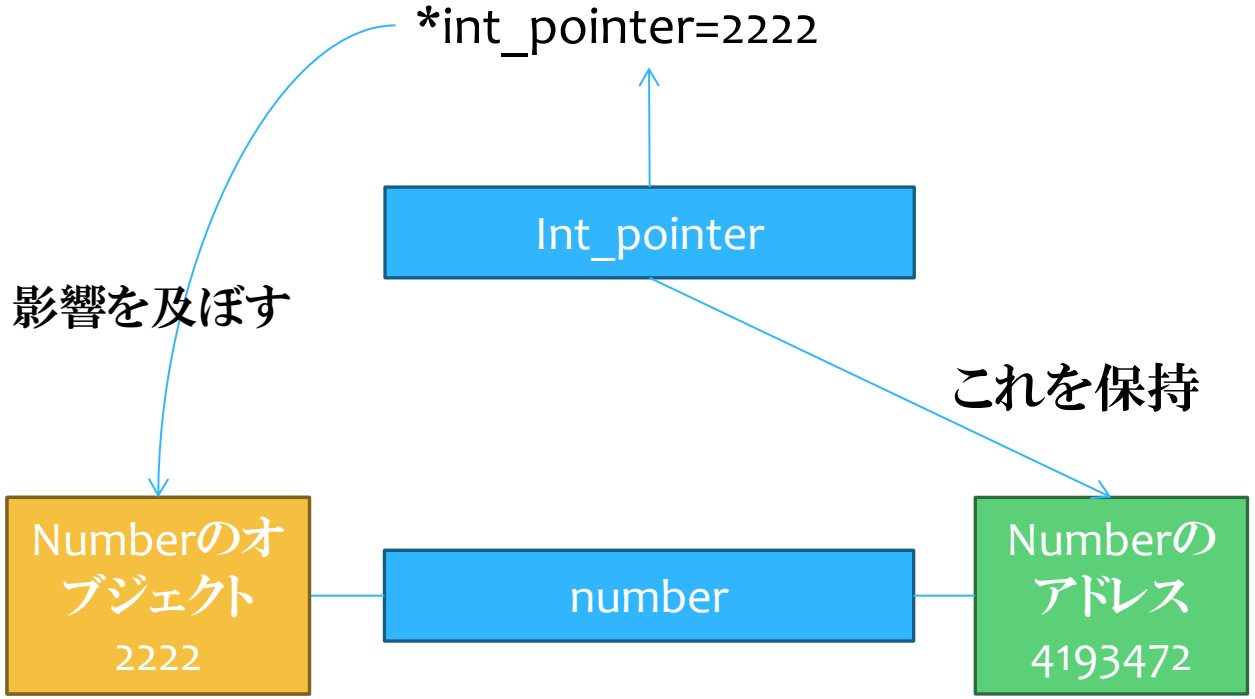
* ここで整理

int_pointer=&numberのとき



address	object
…	
4193472 (number)	1111
…	
1000	
1001 (int_pointer)	(4193472)
1002	
…	

ポインター型演算子



address	object
…	
4193472 (number)	2222
…	
1000	
1001 (int_pointer)	(4193472)
1002	
…	

printf(“%d”,number)の値も2222に変化する

ポインター型演算子

int　　*int_pointer;

この時、int_pointerはアドレス番地を示す

初期化をしていない状態なので、参照しようとするとエラーが返ってくる。

(NullPointerError)

→ぬるぽです。

address	object
…	
1000	
1001 (int_pointer)	
1002	
…	

ポインターと関数

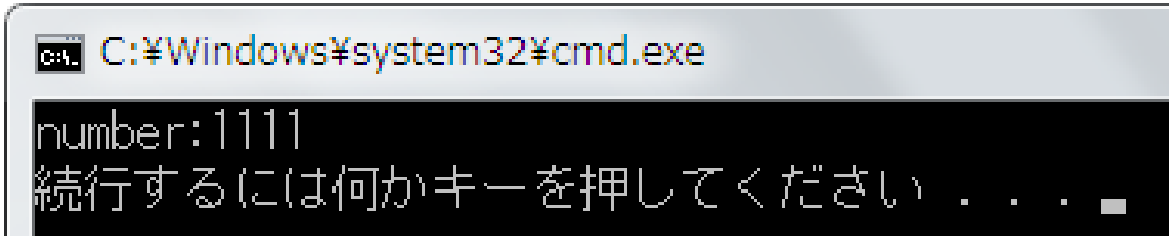
```
int test(int value){
        value - =111;
        return value;}
int main(){
        int number=1111;
        test(number);
        printf("number:%d¥n",number);
         return 0;}
```

address	object
…	
(number)	1111
…	

この時のprintfの値は…？

ポインターと関数

printf("number:%d¥n",number);



address	object
…	
(number)	1111
…	

関数の中で変えたらmain文の中でも変えたい！！

Q1　先ほどのコードで実装してみよう！

ポインターと関数

```
int test(int value){
        value - =111;
        return value;}
int main(){
        int number=114514;
        test(number);
        printf("number:%d¥n",number);
         return 0;}
```

ポインターと関数

関数の中で変えたらmain文の中でも変えたい！！

ではどうしたらいいか……？

ポインターと関数

ポインタに→ void test(int* value){
中身参照→ *value-=111;
;}

int main(){
int number=1111;
アドレス渡し→test(&number);
return 0;}

address	object
…	
(number)	1111
…	

ポインターと関数

test(&number);

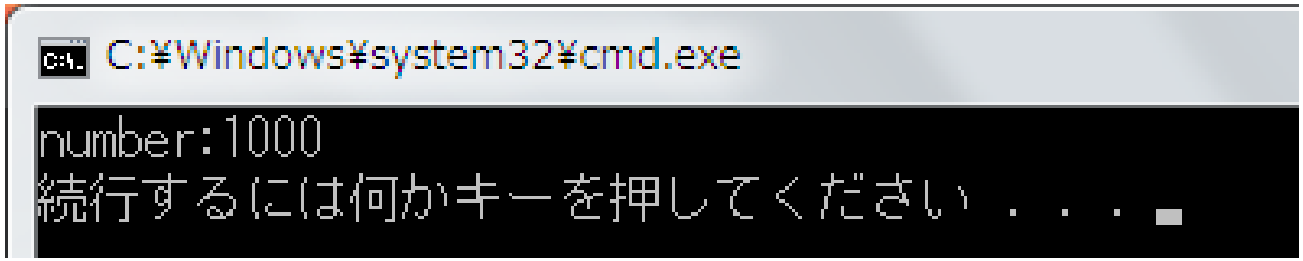
これを実行するとint* value=&number;

のように行われて、関数内でnumberを処理していくことになる！

address	object
…	
(number)	1111
…	

test(&number);実行後

address	object
…	
(number)	1000
…	



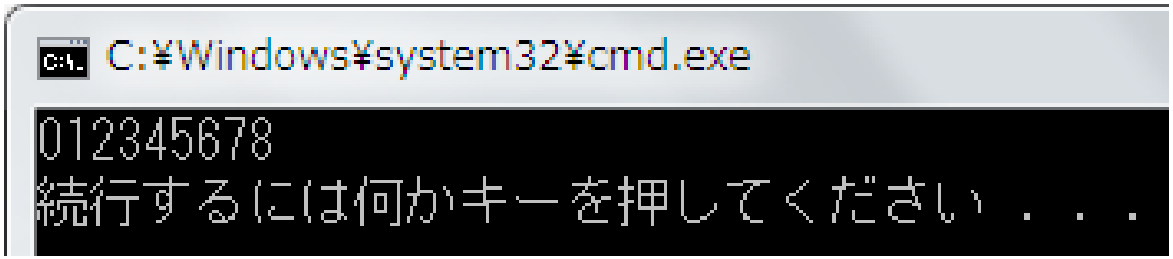
ポインターと配列

* C言語講座ですでに学んだ配列は覚えてますか？

  例)int numbers[9]={0,1,2,3,4,5,6,7,8};

  次のコードを実行させてみましょう。

```
int numbers[]={0,1,2,3,4,5,6,7,8};
for(int i=0;i<9;i++)
        printf("%d",numbers[i]);
printf("¥n");
```

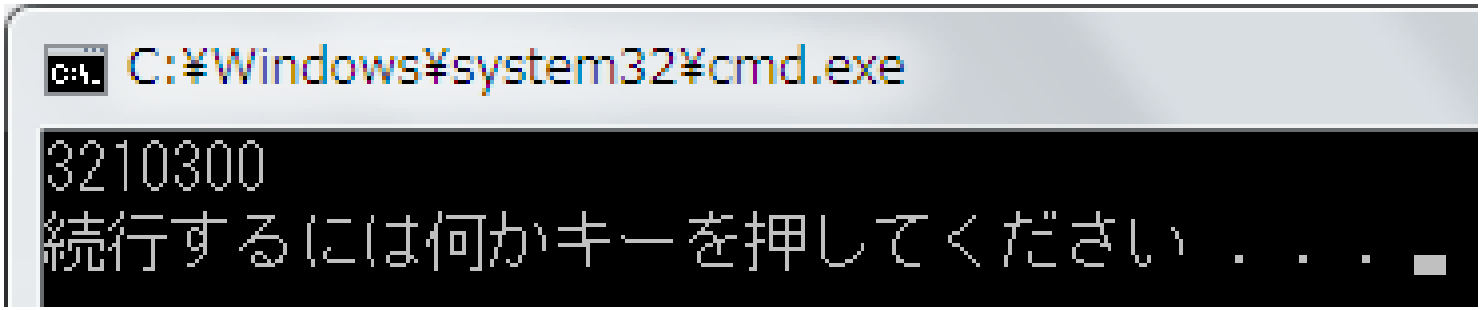


ポインターと配列

* ここで、

```
printf("%d",numbers);
```

これを実行させてみると以下のようになる。



そう、みんな大好き**ポインターちゃん**です。

配列のポインター的利用

printf(“%d”,numbers);

* これはnumbersの先頭アドレスを示す。

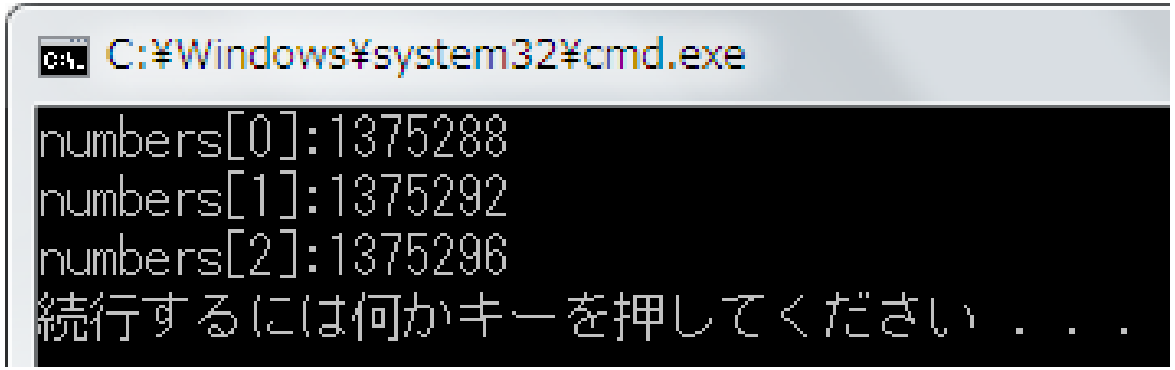
address	object
…	
(numbers[0])	0
(numbers[1])	1
(numbers[2])	2
…	
…	
…	

ポインターと配列

表現の都合上numbersのサイズを3へ減らさせていただきました。

* ポインターならアドレスを見てみよう！

```
for(int i=0;i<3;i++)
        printf("numbers[%d]:%d¥n",i,&numbers[i]);
```



アドレスはコンパイルするごとにメモリの場所が変わるので数値がころころ変わります。

ポインターと配列

* int numbers[]={0,1,2};

これを定義すると右表のように連続して
メモリが確保される。配列とは続いたメモリ
上に存在するものだったのだ！

address	object
…	
1375288 (numbers[0])	0
1378292 (numbers[1])	1
1375296 (numbers[2])	2
…	
…	
…	

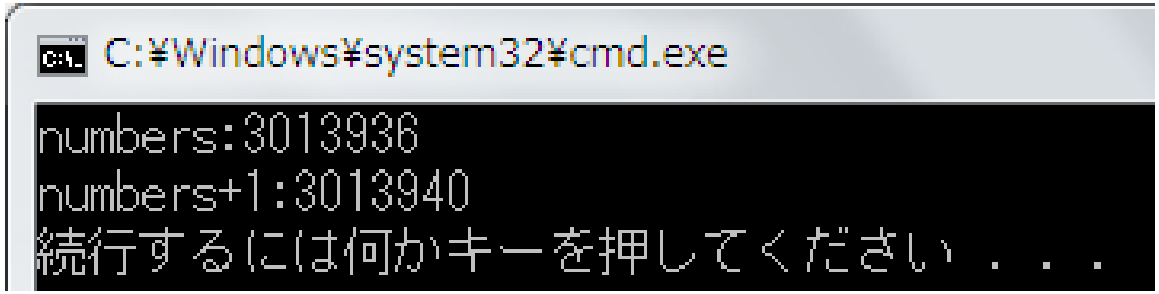
※int型のアドレス単位は4なので4つずつ増加している。
詳しく聞きたい方は終了後お近くの先輩へ！

配列のポインター的利用

* ではnumbersのポインターを一つずらせばnumbers[1]のアドレスを示せるの…？

```
printf("numbers:%d¥n",numbers);
printf(“numbers+1:%d¥n”,numbers+1);
```

address	object
…	
(numbers[0])	0
(numbers[1])	1
(numbers[2])	2
…	
…	
…	

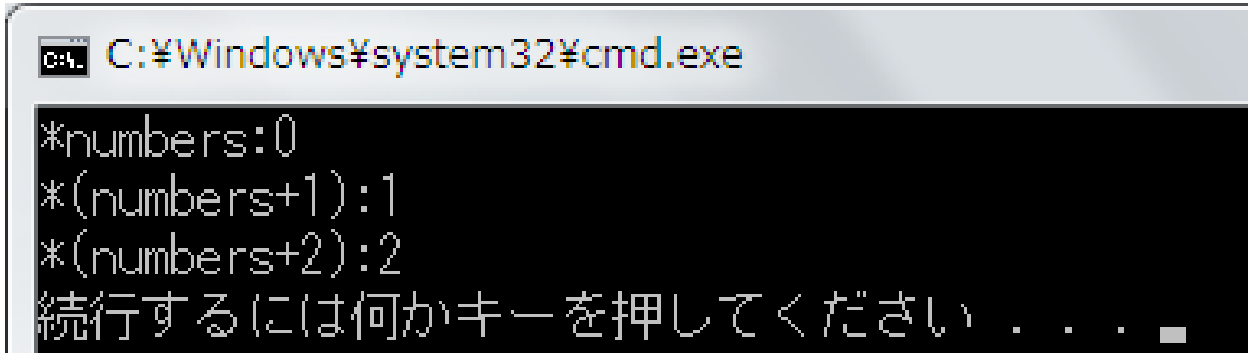


※ポインタ型変数+1とやるとアドレス単位だけ次へ進んでくれます。

配列のポインター的利用

* 示せるのなら中身も見られるのか…？

```
printf("*numbers:%d¥n",*numbers);
printf("*(numbers+1):%d¥n",*(numbers+1));
printf("*(numbers+2):%d¥n",*(numbers+2));
```



address	object
…	
(numbers[0])	0
(numbers[1])	1
(numbers[2])	2
…	
…	
…	

配列のポインター的利用

* つまり、配列α[β]があるとすると

  α[β] と *(α+β) は同義である！

* だからどうした？？となりますよね、でもこれ大切なんです。これが必要な場合は応用編での解説へ続く。

関数とポインターと…？

注意事項

* ここから先は応用になります。
* ここまでの資料でポインターの基本的な概念の説明は終了になります。
* ここから先はがっつり聞かないでも

「あ^～、こんなのあるのか～」

程度の知識で十分です。

関数とポインターと…？

* C言語では複数の値を返す関数は作れない！！
* 例)int[] test(int value)

* ではどうすれば複数の値を返せるのか…？

関数とポインターと…？

* 目的：ある数を受け取ったらその数、その数の二乗、その数の三乗を返す関数を作りたいな～。
  でも関数って複数の値返せないなぁ～。

複数の値を返す関数

* 手段: 複数のアドレスを受け取って、そのアドレスに書き込めば疑似的に複数の値を返すことができる。

```
void power(int value,int* address1,int* address2){
        *address1=value*value;
        *address2=value*value*value;
};
```

複数の値を返す関数

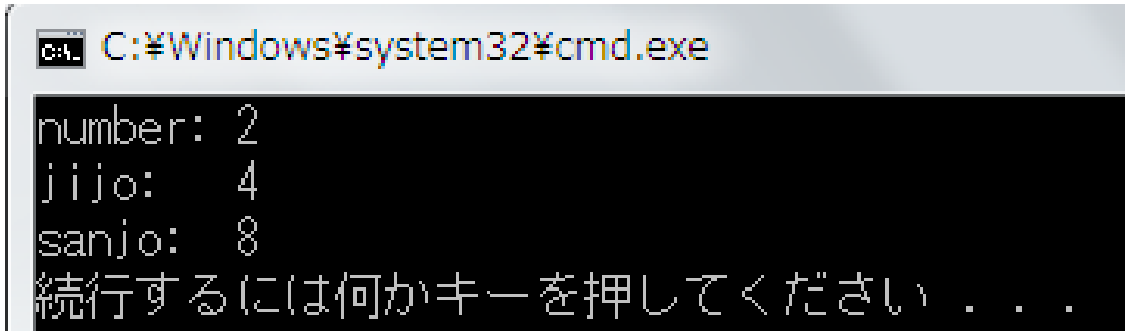
* main文

```
int number=2;
int jijo,sanjo;
power(number,&jijo,&sanjo);
```

address	object
(number)	2
(jijo)	
(sanjo)	

power実行後

address	object
(number)	2
(jijo)	4
(sanjo)	8



関数とポインターと…？

* 目的: 関数使って配列の順番を自動的に動かしたいな～。でも関数内で動かしてもmain文では意味ないだろうからなぁ～。

* あ、ならポインターでやればいいじゃん！

  と、なりますが配列をポインターとして関数には渡せません。

関数とポインターと配列と

* 手段: 関数で配列を受け取ってうまいことやっちゃう。

```
void swap(int array[]){
        for(int i=0;i<(4/2);i++){
                int temp=array[i];
                array[i]=array[3-i];
                array[3-i]=temp;
        }
};
```

今回arrayサイズは4に行うので、全部入れ替えるのに回数は2回必要

関数とポインターと配列と

main文

```
int numbers[4]={0,1,2,3};
swap(numbers);
```

```
numbers[0]:0
numbers[1]:1
numbers[2]:2
numbers[3]:3
swap実行...
numbers[0]:3
numbers[1]:2
numbers[2]:1
numbers[3]:0
```

address	object
(numbers[0])	0
(numbers[1])	1
(numbers[2])	2
(numbers[3])	3

swap実行後

address	object
(numbers[0])	3
(numbers[1])	2
(numbers[2])	1
(numbers[3])	0

関数とポインターと配列と

何故swapはポインタではないのにメイン文でも変更が行われたのだ……？

→配列はポインタの集まりなので関数で変更するとメイン文にも影響する！

α[β] と *(α+β) は同義である！

これを忘れて設計すると思わぬエラーが発生するため注意しましょう。

address	object
(numbers[0])	0
(numbers[1])	1
(numbers[2])	2
(numbers[3])	3

swap実行後

address	object
(numbers[0])	3
(numbers[1])	2
(numbers[2])	1
(numbers[3])	0

演習問題1

* int型の変数 K3を作成し、k3のアドレスをポインターPK3に読み込ませる。変数K3の値は好きな値を入力してください。

* PK3に読み込ませた後、PK3を用いてk3の数値を2倍にして出力する。

演習問題2

* int型の変数 K3を作成する。変数K3の値は好きな値を入力してください。

* 引数を二倍にする関数を作成する。ただし引数をポインターにしてください。

演習問題3

* 大きさが8のInt型配列numbersを作成し、numbersに数値を入力する。

* Int型のポインターを引数とし(このとき引数は1つとする)、引数から+2番目までのすべての要素を二倍にする関数を作成する。
* 例:numbers[3]~numbers[5]までを二倍にする

※関数への引数二通りくらい試してみてください

※配列の大きさを超えないように注意してください