プログラミングI
数理物理，総合理学等向け

2019年1月7日

海谷 治彦

目次

- 11章[レ] 10章 [明] ポインタ
  - C言語の最大難関といわれています･･･orz
- コンピュータ内の情報表現 (復習)

- 演習の解答例 時間があれば

復習かも

バイト，ビット

- デジタルコンピュータの最小情報単位はbit (ビット)と呼ばれ，二進数の一桁に相当する.
  - 例えば，たった1bitでも，「ある人の生死」，「受験の合否」等の情報を表現することができる.
- 数字や文字等，複雑な情報は，この1bitを複数個，束ねて表現する.
- 8bitを束ねた情報を1byte (1B 1バイト)と呼び，わりと広く使われている.
  - 例えば，英数文字の一文字は1Bで表現できる.
- コンピュータのメモリは，この1B単位の情報を格納できる配列みたいなものである. (後述)

二進法	十六進法
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

バイト表現と16進法

- 1Bは8bitなので，原則，0/1の列を8個かかないといけない．
- これはかなり面倒かつ読みにくいので，1Bは通常，16進法の2桁で表現する．
- 対応表は左記の通り．
- 例えば，文字の M は，

0100 1101 なので，

4D と表現できる．

復習

対応表 (ASCII code)

上位ビット→ 下位ビット↓	0000	0001	0010	0011	0100	0101	0110	0111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAC	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF/NL	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	¥	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

例えば,
文字 D は
 0100 0100
文字 3 は
 0011 0011
とコンピュータ内部では
表現されるのが普通.

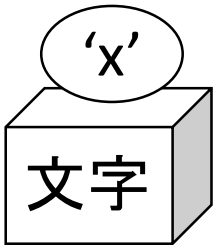
復習++

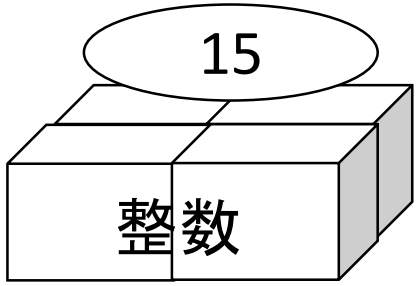
参考 箱の大きさ

- 型によって一個の変数の大きさは異なる.
  - char を1箱(1B)とすると,
  - int 4B分
  - float 4B分
  - double 8B分

が一般的.

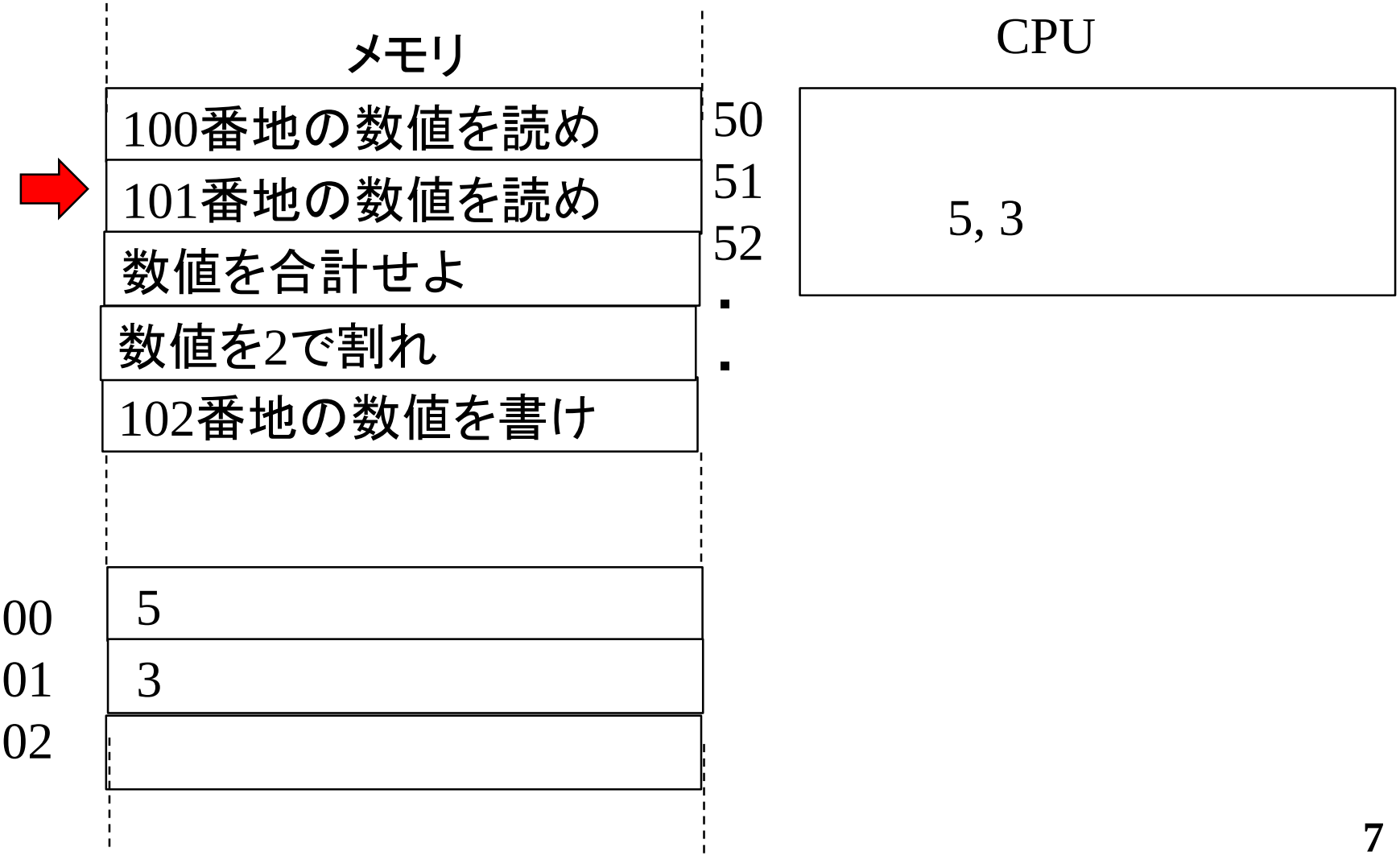
- しかし, C言語にはこの辺の明確な規定が無い.





復習

簡易な例題 ～ 二値の平均



メモリは配列みたいなもの

- 変数が格納されるコンピュータのメモリは配列みたいなものである.
- アドレスとは，この配列の添え字である.
  - 市販のPCで，「メモリ8G」とかいってるのは，このアドレスの上限値でもある．8G≒80億個
- アドレスは通常，「〇〇番地」という単位で呼ばれる.
  - ニホンゴ訳そのまんまだな
- アドレスは，通常16進数で表現する.
- 一つのアドレスが指すメモリに，に1バイトのデータが格納できる.
- プログラム中の変数をメモリ中のどこに配置するかは，ccコマンド(とwindows等のOS)が決める.
- 型によって，必要な変数のサイズが異なるため，一つの変数を複数要素に連続して格納する.

前々頁に対応するプログラム

- 三つの変数が，メモリ中に4個おきに配置されているのがわかる.
- intは4B必要なため，4個おきとなっている.

```
sh-3.1$ ./a.exe
5 at 0028FEE4, 3 at 0028FEE8, 4 at 0028FEEC
sh-3.1$
```

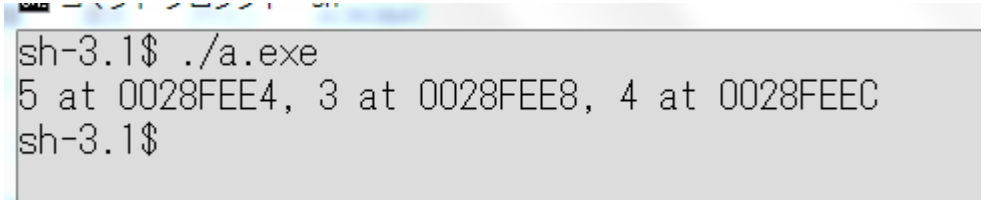
```
// adr.c
#include <stdio.h>

int main(void){
int a102, a101=3, a100=5;
  a102 = (a100+a101)/2;
  printf("%d at %p, %d at %p, %d at %p¥n",
    a100, &a100, a101, &a101, a102, &a102);
  return 0;
}
```

番地	内容
0028FEE2	
0028FEE3	
0028FEE4	5
0028FEE5	
0028FEE6	
0028FEE7	
0028FEE8	3
0028FEE9	
0028FEEA	
0028FEEB	
0028FEEC	4
0028FEED	
0028FEEE	
0028FEEF	

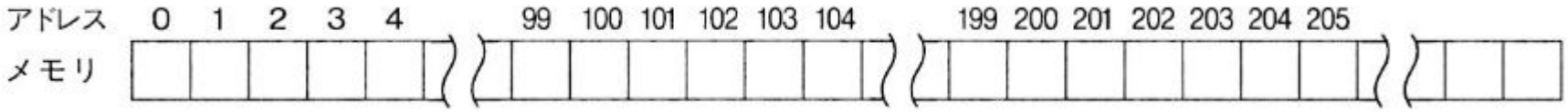
ポインタとは何か？

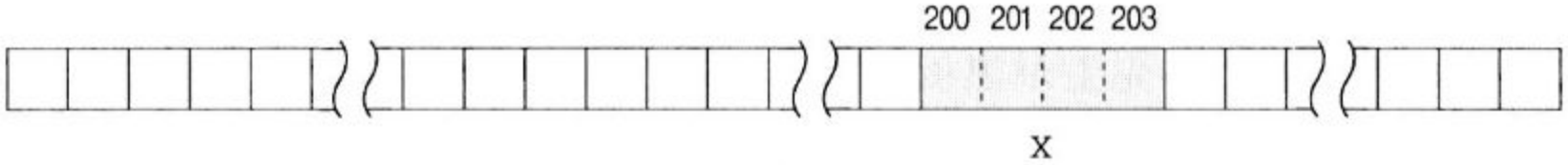
- 変数の配置されているアドレスを記録する変数である.
- メモリを配列とみたてれば，その添え字(index)を記録する変数である.
- 具体的には，前頁の「0028FEE4」等の値を記録する変数である.



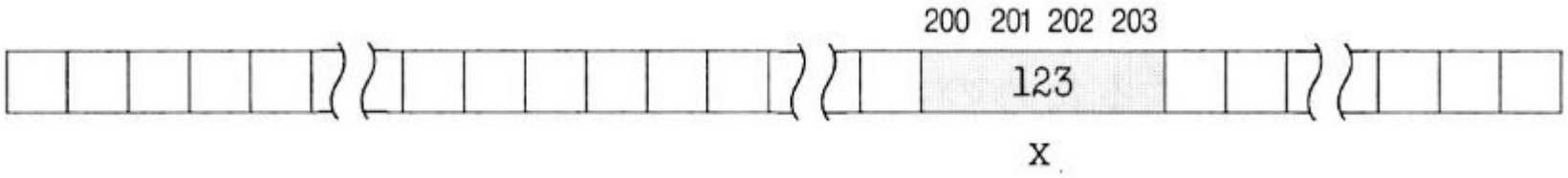
- ポインタの最小値は0である，負は無い.
  - 配列のようなものの添え字なので負は無い.
- ポインタの上限はCPU，実メモリ，OS等に依存するが，大体，以下な感じ.
  - 42億 (32bitマシンの場合 2^{32} 個)
  - 18×10^{18} (64bitマシンの場合 2^{64} 個)

教科書のイメージ





int 型が４バイトであるコンパイラを想定している



ポインタで何が嬉しい？

- コンピュータのメモリを直接操作してる雰囲気を味わえて面白い(人もいるかもしれない).

- 関数の引数を関数内で更新することが可能となる.
- より一般的には，異なるスコープにある変数にアクセスするための手段を提供する.
  - メモリはプログラム全体で共有されている配列のようなもの.
- 配列を走査(scan)するのに便利.
- 変数を事前に宣言せずに，プログラムの実行中に適宜，確保するのに役立つ. (本授業では扱いません)

復習

変数，スコープ，ブロック

- 変数を使うには宣言しなければならない.
- 宣言が有効なのは，宣言を行ったブロックの内側のみである.
- ブロックとは { と } で囲まれた部分である.
- 前頁の例もあるが，別にmainの先頭でなくても，ブロックの先頭で宣言できる.
  - 実は main の外でもできるのだが.
- 変数が利用可能な範囲をスコープ (Scope)と呼ぶ.

- 最も外側のブロックで宣言すれば全体から見えるが，それはあまりお勧めしない方法である.
- 最小スコープで変数を宣言するのが良いとされる.

復習

List 6-4 改

```
// list6-4x.c
#include <stdio.h>

int main(void){
  int i=0; // 行を数える
  while(i<10){ // 行を書くループ
  int j=0; // *の数を数える
    printf("%d ", i);
    while(j<i){ // *を書くループ
      printf("*");
      j++;
    }
    printf("¥n");
    i++;
  }
  return 0;
}
```

```
bash-3.1$ cc list6-4x.c
bash-3.1$ ./a
0
1 *
2 **
3 ***
4 ****
5 *****
6 ******
7 *******
8 ********
9 *********
bash-3.1$
```

復習

段付け(indent)をしよう

- ブロックの開始と終了，if や while の入れ子の見た目が分かりやすいように，プログラムの段付けをしてください！
- 右の例を参照.
- 基本，{} があれば，その内側は一段下げる.
  - 空白文字かタブ
- 多くの受講生は段付けをしてないので，プログラムが見難くない？

```
// list6-4x.c
#include <stdio.h>

int main(void){
  int i=0; // 行を数える
  while(i<10){ // 行を書くループ
  int j=0; // *の数を数える
    printf("%d ", i);
    while(j<i){ // *を書くループ
      printf("*");
      j++;
    }
    printf("¥n");
    i++;
  }
  return 0;
}
```

ポインタに関する操作 (概要)

- ポインタはアドレスを記録する変数である.
- よって，定義，代入，参照，演算を通して利用する.
- 定義
  - int *p; 等
- 参照
  - printf中では %p, %x が普通. %d でもいいけどお勧めしない.
- 代入
  - 普通に = で代入
- 演算
  - &と*
  - 四則演算は足し算と引き算のみ. *, /, % 等は行えない.
  - + は1個先のアドレスに進む.
  - - は1個前のアドレスに戻る.
  - 進む/戻る個数はデータ型に依存.

ポインタの定義

- 通常の変数同様，ブロック冒頭で定義を行ってから，ポインタを利用する．
- ポインタはアドレスを保持する変数であるが，単にアドレス値だけでなく，保持されているデータの型も定義時点で指定する．
- よって，例えば，int型の値が保持されるアドレスを記憶するポインタは，

```
int *p;
```

  等で宣言する．(後述の +， - 適用のため)

- 気持ち的には int* で一つの型(intが在るアドレス)という感じだが，一般に `int* p` とは書かない．
  - 文法的な誤りとはならないが．
- 教科書にもあるが，`int *p, *q;` と複数のポインタを同時に宣言できる．

ポインタの参照

- ポインタの値はint等と同様に参照できる.
- printf 等では, %d や %f の代わりに %p を用いるのが普通.

- C言語の学習ではポインタの値を表示することがよくあるが, 現実のプログラムではほぼ無い.
  - 上記の %p もあまり使わない.
  - デバッグ(プログラムの欠陥除去)等の目的くらい.
- 後述の演算子の適用や代入において, 参照が用いられる.

ポインタの演算 1/2 & と *

- ある変数が配置されているアドレスを知るための演算子として，& がある.
- 気分的には，配列要素から，添字を知るための演算子みたいな感じ.

  (配列には，そんな機能は無いけど)
- & によって，ある変数が配置されているアドレスをとりだし，代入でポインタに記録することができる.
  - &によって，メモリという配列の添え字を取り出す感じ.
- 逆に，あるアドレスにある値を示す演算子 * がある.
- *があることによって，ポインタpに保存されている値を参照したり更新したりすることができる.

Fig. 11-6

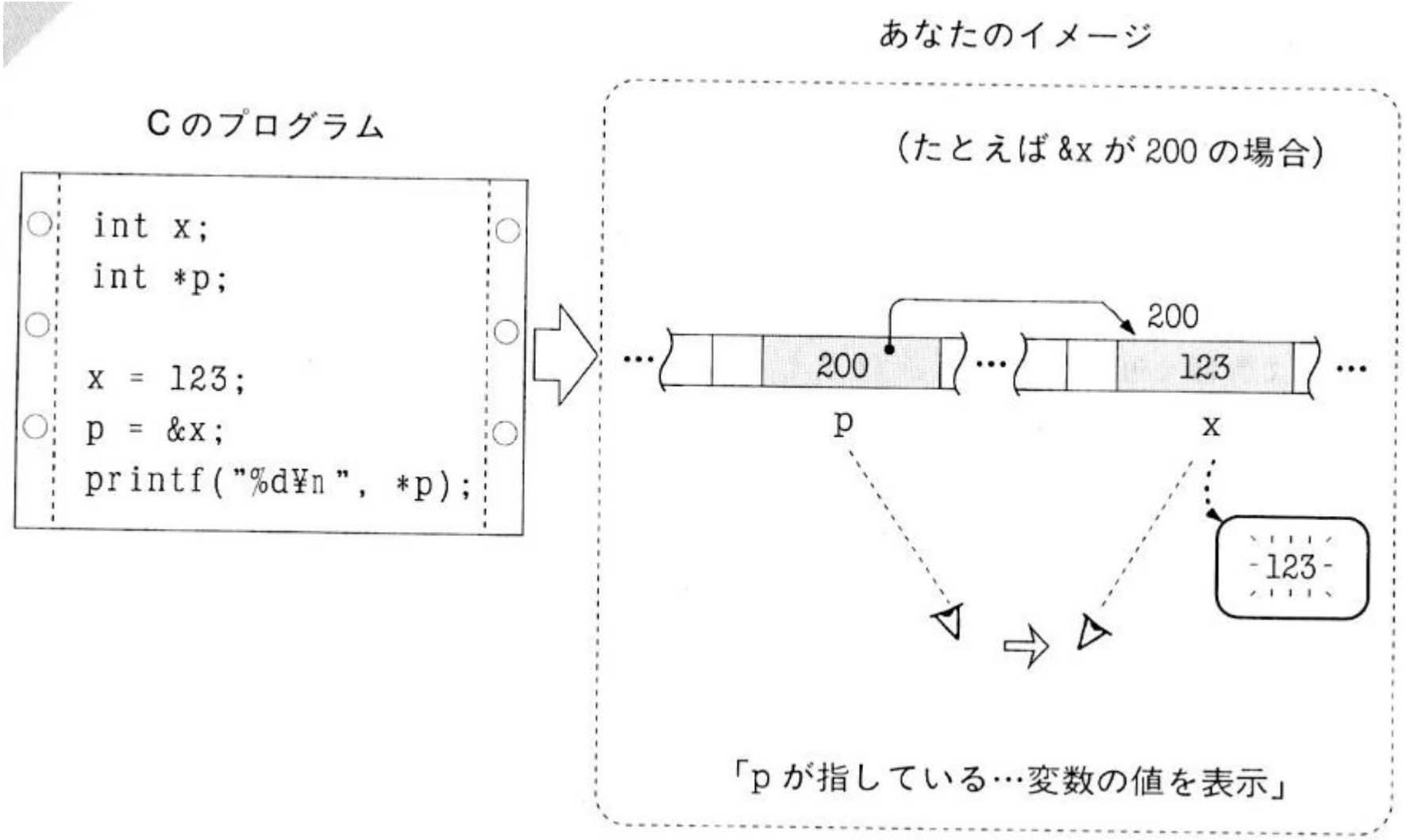
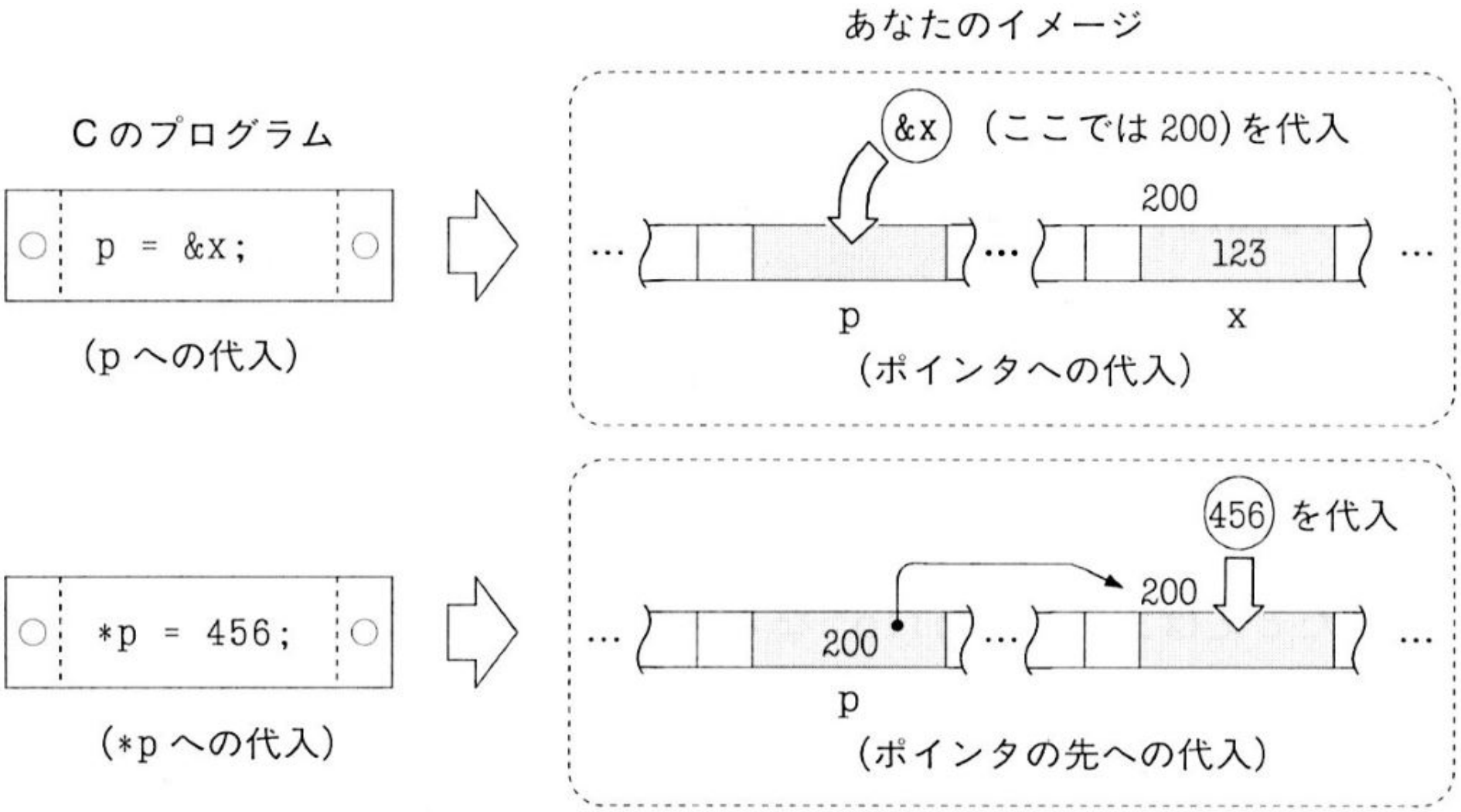

あなたのイメージ
Cのプログラム
int x;
int *p;
x = 123;
p = &x;
printf("%d¥n", *p);
(たとえば &x が 200 の場合)
200
200
123
p
x
123
「p が指している…変数の値を表示」

Fig. 11-7


あなたのイメージ
Cのプログラム
p = &x;
(pへの代入)
&x (ここでは200)を代入
200
123
p
x
(ポインタへの代入)
*p = 456;
(*pへの代入)
456 を代入
200
200
p
(ポインタの先への代入)

Scopeに関するちょっとした例

```
// scope2.c
#include <stdio.h>

int main(void){
int a=10; // ココで定義した変数 a と
int *p;
  p=(&a); // 上記のaのアドレスをpに保存
  printf("%d at %p¥n", a, &a);
  if(a==10){
  int a=22; // ココで定義した a は,
    (*p)=12; // 直上のaではなく,
             // main直下のaを更新
    printf("%d at %p¥n", a, &a);
  }
  printf("%d at %p¥n", a, &a);
  // 別物であることがアドレスを見るとわかる
  return 0;
}
```

```
sh-3.1$ ./a.exe
10 at 0028FEE8
22 at 0028FEE4
12 at 0028FEE8
```

アドレス	値
0028FEE3	
0028FEE4	内側の a 22
0028FEE5	
0028FEE6	
0028FEE7	
0028FEE8	外側の a 10→12
0028FEE9	
0028FEEA	
0028FEEB	

ポインタの演算 2/2 + と -

- ポインタには整数を足すこと+と引くこと-ができる.
  - 他はできない！
  - ポインタ同士を足す/引くこともできない.
- 足し算は，足した個数分，先のアドレスに進める，を意味する.
- 引き算は逆.
- ポインタの型によって，アドレスの進む数が異なる.
  - 通常，int * なら4個毎，char * なら1個毎に進む.

- 配列は要素が連続して配置されていることが保証されるので，通常，ポインタの足し算，引き算は配列と一緒に用いられる.

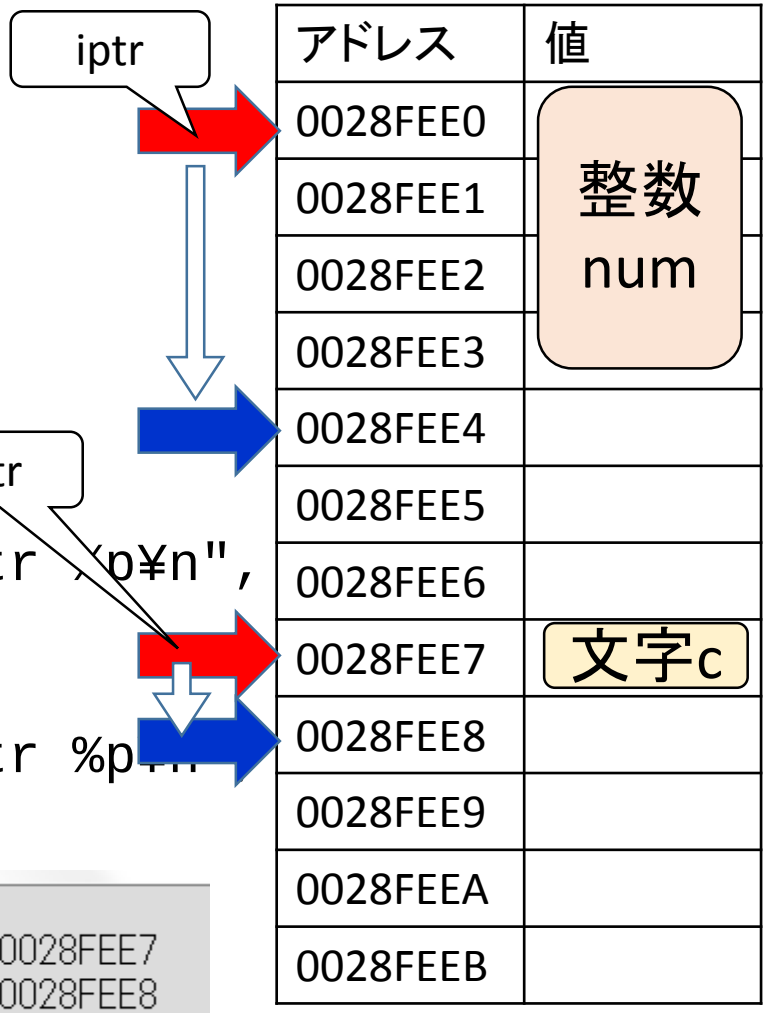
char*とint*で++の進み具合が違う

```
// inc1.c
#include <stdio.h>

int main(void){
char c='x', *cptr;
int num=11, *iptr;
  iptr=(&num);
  cptr=(&c);
  printf("int ptr %p,  char ptr %p¥n",
   iptr, cptr); // 赤矢印
  iptr++; cptr++;
  printf("int ptr %p,  char ptr %p¥n",
   iptr, cptr); // 青矢印

  return 0;
}
```

```
sh-3.1$ ./a.exe
int ptr 0028FEE0,  char ptr 0028FEE7
int ptr 0028FEE4,  char ptr 0028FEE8
sh-3.1$
```



配列との組み合わせ

```
// inc2.c
#include <stdio.h>

int main(void){
char c[]="Hello", *cptr;
int i=0, num[]={3, 1, 4, 1, 5}, *iptr;
  iptr=(&num[0]);
  cptr=(&c[0]);
  while(i<5){
    printf("%d at %p, %c at %p¥n",
        *iptr, iptr, *cptr, cptr);
    iptr++; cptr++; i++;
    // iptr = iptr + 1; でもOK
  }
  return 0;
}
```

```
sh-3.1$ ./a.exe
3 at 0028FEC8, H at 0028FEDE
1 at 0028FECC, e at 0028FEDF
4 at 0028FED0, l at 0028FEE0
1 at 0028FED4, l at 0028FEE1
5 at 0028FED8, o at 0028FEE2
```

cptr

アドレス	値
0028FEDD	
0028FEDE	H
0028FEDF	e
0028FEE0	l
0028FEE1	l
0028FEE2	o
0028FEE3	¥0
0028FEE4	
0028FEE5	
0028FEE6	
0028FEE7	
0028FEE8	

iptrも同様だが，
大きいので省略

ポインタの初期化

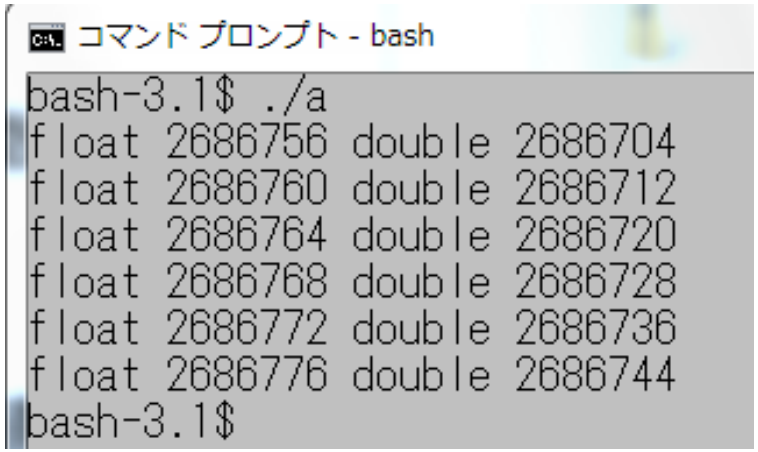
- 前述のように，intやdoubleと同様に代入できる.

- 変数が在るアドレスの値はプログラムの実行の度に変化する可能性がある.
- よって，ポインタにアドレスを定数で代入することはほとんど無い.
- 代わりに，演算子 & をもちいて，アドレスを検索し，その値を代入して初期化するということが一般的.

演習13

- dotcampusからダウンロードできるプログラム ptr1.c は，float と double の配列要素のアドレス値を整数で画面に表示している.
- 少なくとも大学のPCでは，アドレスの増加する数が，float と double で異なる.
- その理由を説明せよ.
- 以下は私のPCの場合の実行例，他のPCでは値が異なる.

4つ毎増加



8つ毎増加

```
/*
 * ptr1.c 演習13用のサンプル
 */
#include <stdio.h>

int main(void){
float dataf[]={3, 1, 4, 1, 5, 9};
double datad[]={3, 1, 4, 1, 5, 9};
int i;
  for(i=0; i<6; i++){
    printf("float %d double %d¥n", &dataf[i], &datad[i]);
  }
  return 0;
}
```

以上