プログラミング基礎
第10回

藤江 真也

2021年6月25日

準　備

- ファイル（0625.tgz）をダウンロード

```
$ wget http://sites.fujielab.org/ip/files/0625.tgz
```

- ダウンロードしたファイルを展開

```
$ tar zxvf 0625.tgz
```

- 展開されたディレクトリに移動

```
$ cd 0625
```

- char.cなどのファイルがあることを確認

```
$ ls
```


文字・文字列

文字（character）

- 人が読める記号
- 言語によって利用する文字は変化
  - 英語：アルファベット
  - 日本語：平仮名，カタカナ，漢字
  - 韓国語：ハングル文字
  - …

สวัสดี Hello こんにちは 안녕하세요 您好 здравствуйте

- コンピュータでは基本的にASCII文字（アルファベット＋いくつかの特殊文字）を扱う
  - 種類はそれほど多くない（100種類程度）

コンピュータは文字を（直接）扱えない！

- 如何なる文字も，数値として扱う
  - 何番目の文字か，ということ→ASCIIコード

下位 4ビット ＼ 上位 4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NU	DL	SP	0	@	P	`	p				−	タ	ミ		
1	SH	D1	!	1	A	Q	a	q			。	ア	チ	ム		
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		
6	AK	SY	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	I	y			ゥ	ケ	ノ	ル		
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	GS	−	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	RS	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	US	/	?	O	_	o	DL			ッ	ソ	マ	゜		

例えば「C」は，16進数で 0x43 番目

文字も数値

- 文字は0～255の数値のいずれかで表現される
  - ASCII文字以外に限る
- **char型**
  - 1バイトの整数値を表す型 であると同時に
  - 1文字を表す型 でもある

例1

char.c

```
#include <stdio.h>

int main (void) {
  char c = 'C', n = 0x43;

  printf ("c = %c, c = 0x%02x¥n", c, c);
  printf ("n = %c, n = 0x%02x¥n", n, n);

  putchar (c);
  putchar (n);
  putchar ('¥n');

  return 0;
}
```


確認

- （確認1）char.c を実行してみよう
  - 実行結果ら，変数cと変数nの中身が同じであることを確認しよう
- （確認2）cやnの値を別の値に変更してどうなるか確認してみよう
- （確認3）cに文字Cを与えるのに ' で囲っているが，これを取り除いたり，" に変えたりしたらどうなるか確認してみよう

文字データ

- char型の変数一つで一つの文字を表す
  （ASCIIに限る）
- 1バイトの整数なので0～255（または−128～127）の数値である
- C言語のプログラム内で直接文字として与えたい場合は，'（シングルクォーテーション）で文字を囲む

```
char c = 'G';
```

GのASCIIコード（0x47）を与えたのと全く同じ意味

文字データの入力・出力

- 文字データを取り扱う関数

出力系：printf，putchar，puts，…
入力系：scanf，getchar，gets，…

などがある．

printfの出力変換指定子による違い

- printf("%c", a);

⇩

変数aのASCIIコードを，文字として出力
「G」と表示される．

- printf("%x", a);

⇩

変数aのASCIIコードを16進数で出力，つまり「47」と表示される．

putchar関数

- 文字を標準出力に書き出す
- 使い方
  - ASCIIコード c が表す文字を標準出力に書き出す場合
    putchar(c);

getchar関数

- 文字を**標準入力**から読み込む
- 使い方
  - 標準入力から読み込んだ文字を c に代入

```
char c;
c = getchar();
```


例2

getchar.c

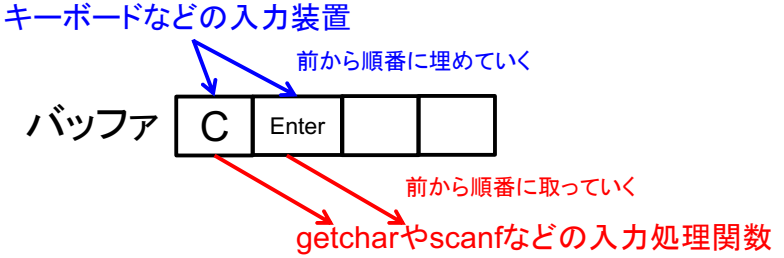
```
#include <stdio.h>

int main (void) {
  char c1, c2;

  printf ("1文字入力してください: ");
  c1 = getchar ();
  printf ("c = %c, c = 0x%02x¥n", c1, c1);

  printf ("1文字入力してください: ");
  c2 = getchar ();
  printf ("n = %c, n = 0x%02x¥n", c2, c2);

  return 0;
}
```


確認

- （確認4）getchar.cを実行してみよう
- （確認5）C「Enter」と入力するとどうなるか確認しよう

バッファ（buffer）

- キーボードやマイクなどの入力装置から得られるデータは一旦バッファと呼ばれる一時記憶に蓄えられる
- 改行（Enter）が入力されるとバッファリングが一旦解かれるが，**Enter自体も文字**として解釈される
  - EnterのASCIIコードは 0x0A（10）



練習

- （練習1）例2（getchar.c）を改良して，Enterを読み飛ばすようにしてみよう
  - 例えば

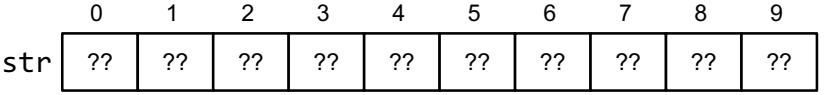
```
1文字入力してください: C ⏎
c = C, c = 0x43
1文字入力してください: D ⏎
c = D, c = 0x44
```

などと出力されるようにする．
（青がユーザ入力）

文字列（string）

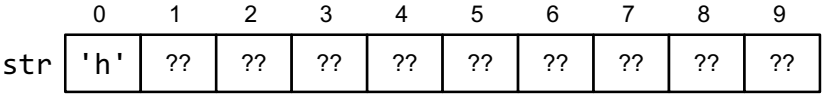
- **文字列**は**文字（char型）**の**配列**

char str[10]; とすると10文字分の配列が用意される



- 各要素に1文字ずつ代入することができる

```
str[0] = 'h';
```



例3

str.c

```
#include <stdio.h>

int main (void) {
  char str[10];
  int i = 0;

  printf ("10文字までの文字列を入力してください: ");
  do {
    str[i] = getchar ();
  } while (str[i++] != 0x0A);

  i = 0;
  printf ("読み込んだ文字列: ");
  do {
    putchar (str[i]);
  } while (str[i++] != 0x0A);

  return 0;
}
```


確認

- （確認6）例3(str.c)を実行してみよう
  - 入力が促されたら hello などと入力してみよう
  - 実行結果から
    - str[i]やstr[i++]がどのような意味を持つか考えよう
    - str[i++] != 0x0A の 0x0A は何を意味するか，考えよう
  - なぜwhile文ではなくdo-while文を使っているか考えよう

文字列の初期化

- **配列**なので，**文字列**も**初期化**できる

```
char a[] = {'A', 'N', 'S', 'I', ' ', 'C' };
```

でもいいが，単純に

```
char a[] = "ANSI C";
```

でもよい．

- **文字列**は " （ダブルクォーテーション）で囲む
- **配列**の**要素数**は**文字数＋1だけ必要**
  - 最後にかならずヌル文字（ASCIIコード0）を入れる必要があるため

例4

str2.c

```
#include <stdio.h>

int main (void) {
  char str1[11] = "C Language", str2[] = "C Language";
  int i;

  for (i = 0; i < 11; i++) {
    printf ("str1[%02d] = %c (0x%02x), str2[%02d] = %c (0x%02x)¥n",
             i, str1[i], str1[i], i, str2[i], str2[i]);
  }

  printf ("printf: %s, %s¥n", str1, str2);

  return 0;
}
```


確認

- （確認7）str2.cを実行してみよう
  - 実行結果から，配列str1と配列str2が同じ内容であることを確認しよう
  - str1[i] のように1文字ずつ取り出すこともできるし，printf("%s", str1); のようにまとめて表示することもできる

ポインタ

例5

pointer.c

```
#include <stdio.h>

int main (void) {
  int v1 = 10;
  int v2 = 20;
  int *ptr;

  ptr = &v1;
  printf ("v1@%p = %d, v2@%p = %d, ptr = %p¥n", &v1, v1, &v2, v2, ptr);

  *ptr = 100;
  printf ("v1@%p = %d, v2@%p = %d, ptr = %p¥n", &v1, v1, &v2, v2, ptr);

  ptr = &v2;
  printf ("v1@%p = %d, v2@%p = %d, ptr = %p¥n", &v1, v1, &v2, v2, ptr);

  *ptr = 200;
  printf ("v1@%p = %d, v2@%p = %d, ptr = %p¥n", &v1, v1, &v2, v2, ptr);

  return 0;
}
```


ポインタ

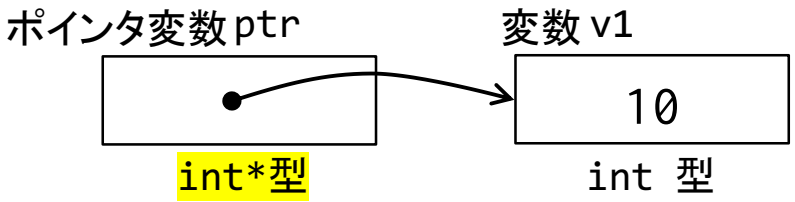
指示棒

レーザポインタ

マウスポインタ

- 一般的に，ポインタとは何かを指すもの
  - point "指す" + er(or) "～する人/物"

ポインタ変数



ポインタ変数の宣言

```
int *ptr;
```

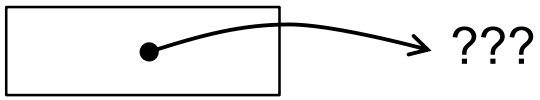
型（*まで含めて型と考えた方が分かりやすい）

変数名

```
int v1=10, v2, *ptr;
```

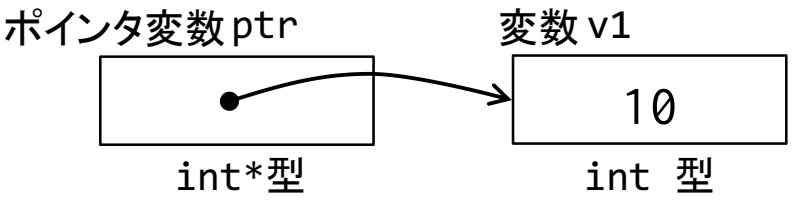
この場合も ptr はint型のポインタ変数
v1 と v2 は int型の変数

- 宣言しただけでは箱ができるだけで指す先は決まらない



ポインタ変数の参照先の設定

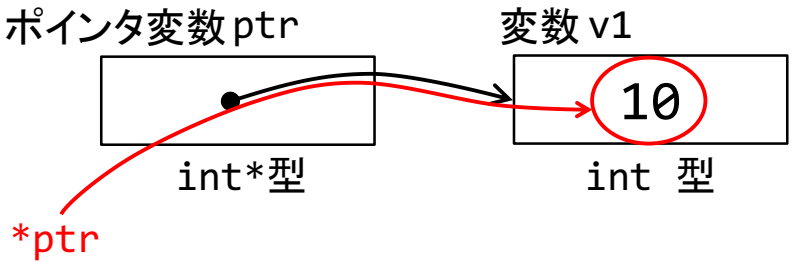
```
int v1 = 10;
int *ptr;

ptr = &v1;
```



ポインタ変数の参照先の利用(1)

```
int v1 = 10;
int *ptr;

ptr = &v1;
printf("%d¥n", *ptr);
```

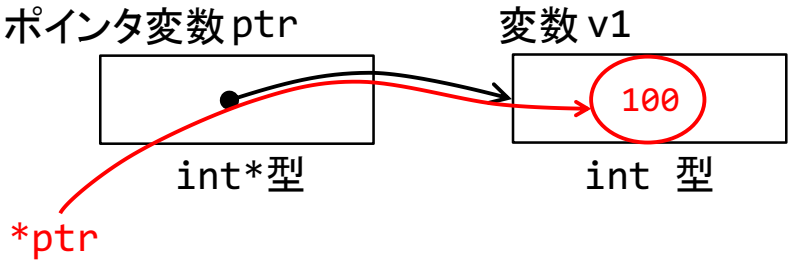
```
printf("%d¥n", v1);
```



ポインタ変数の参照先の利用(2)

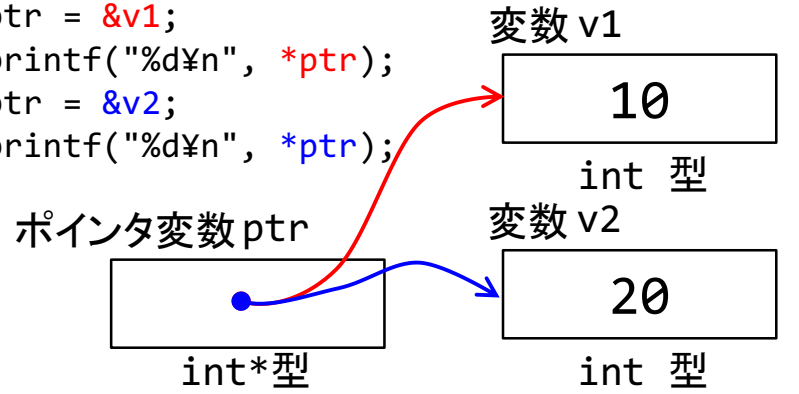
```
int v1 = 10;
int *ptr;

ptr = &v1;
*ptr = 100;
```

```
v1 = 100;
```



ポインタ変数の参照先の付け替え

```
int v1 = 10, v2 = 20;
int *ptr;

ptr = &v1;
printf("%d¥n", *ptr);
ptr = &v2;
printf("%d¥n", *ptr);
```



ポインタ変数の正体

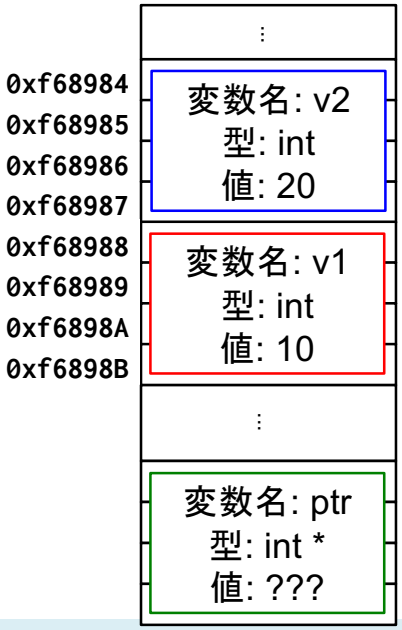
アドレス値！

例5の実行結果

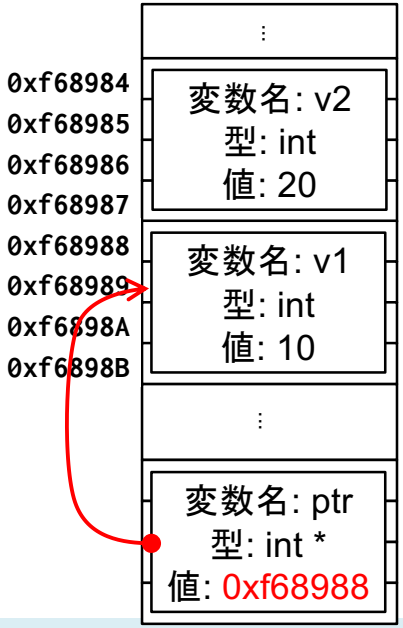
```
ptr = &v1;
v1@0x7fff5cf68988 = 10, v2@0x7fff5cf68984 = 20, ptr = 0x7fff5cf68988

*ptr = 100;
v1@0x7fff5cf68988 = 100, v2@0x7fff5cf68984 = 20, ptr = 0x7fff5cf68988

ptr = &v2;
v1@0x7fff5cf68988 = 100, v2@0x7fff5cf68984 = 20, ptr = 0x7fff5cf68984

*ptr = 200;
v1@0x7fff5cf68988 = 100, v2@0x7fff5cf68984 = 200, ptr = 0x7fff5cf68984
```



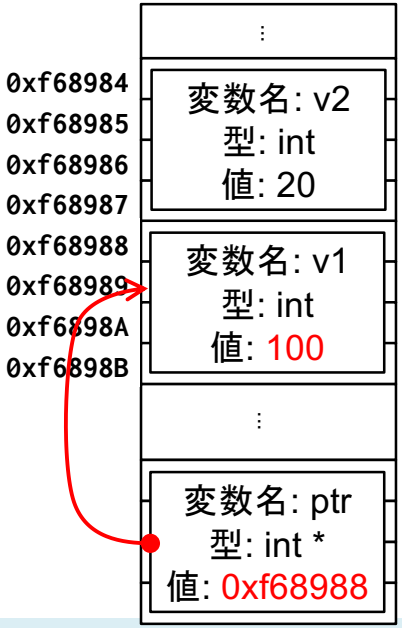
```
int v1 = 10;
int v2 = 20;
int *ptr;
```



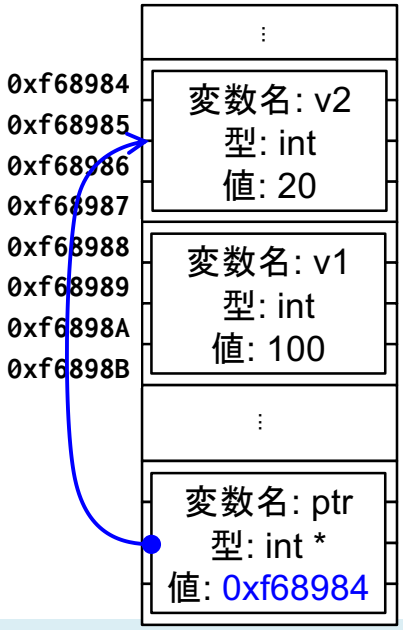

```
ptr = &v1;
```



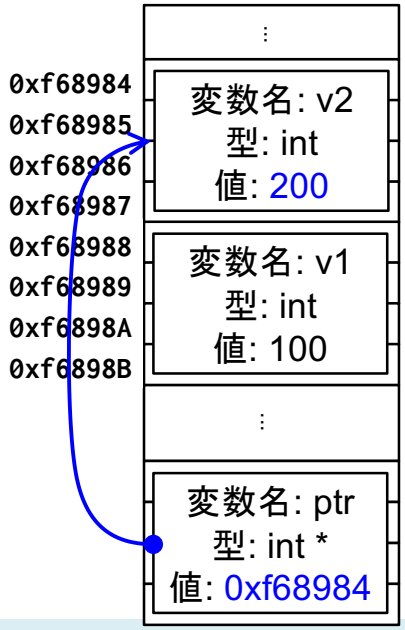

```
*ptr = 100;
```



```
ptr = &v2;
```




```
*ptr = 200;
```



アドレス演算子 & と間接参照演算子 *

- int型を例にすると …

```
int a;
```
int型の変数aの**宣言**

```
int *ptr;
```
int型のポインタ変数ptrの**宣言**

&a ⟷ ptr

int型の変数aの**アドレス**

int型変数の**アドレス**

&は**アドレス演算子**

a ⟷ *ptr

int型の変数aの**値**

ptrのアドレスにある変数の**値**

*は**間接参照演算子**

確認

- （確認8） 例5(pointer.c)のprintfで，*ptrを表示して，参照先の変数の値と同じになっていることを確認しよう．
- （確認9） 例5(pointer.c)でptrの値を強制的に適当な値（例えば0）にしてみよう．7行目で 0 を代入してみればよい．

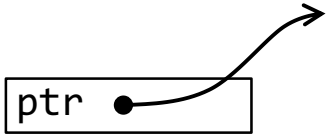
ポインタ変数と配列

- 配列
  ```
  int a[10];
  ```

a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]

ポインタ変数と配列

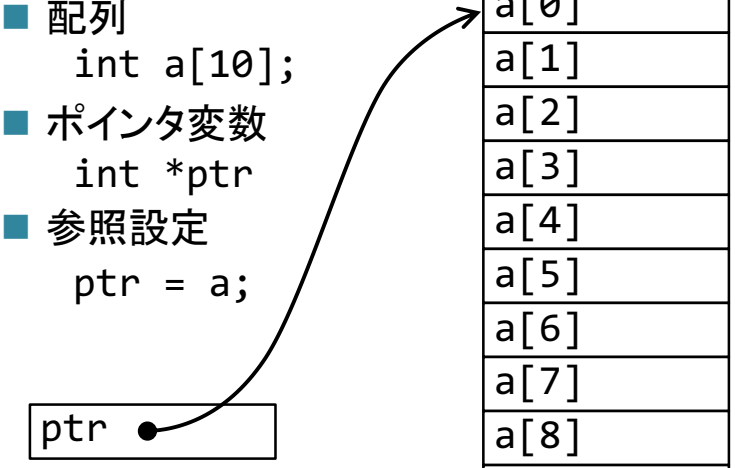
- 配列
  ```
  int a[10];
  ```
- ポインタ変数
  ```
  int *ptr
  ```



a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]

ポインタ変数と配列

- 配列
  ```
  int a[10];
  ```
- ポインタ変数
  ```
  int *ptr
  ```
- 参照設定
  ```
  ptr = a;
  ```

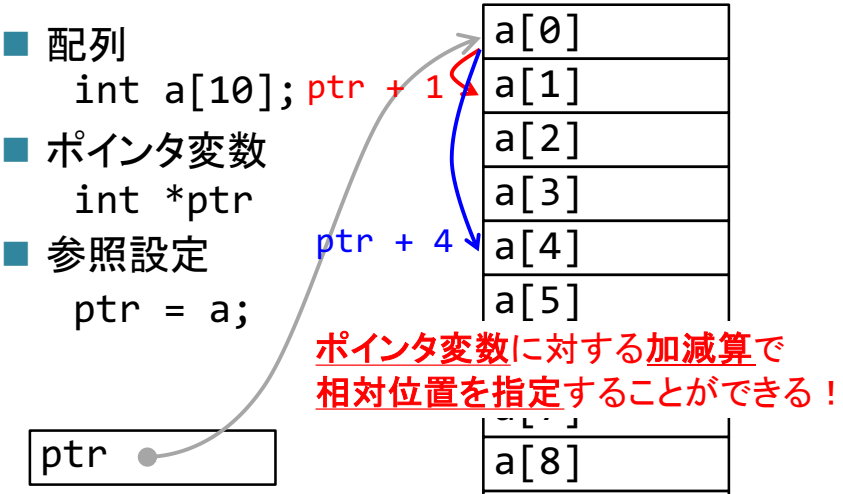


a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]

配列名は先頭要素のアドレス値！

ポインタ変数と配列

- 配列
  ```
  int a[10];
  ```
- ポインタ変数
  ```
  int *ptr
  ```
- 参照設定
  ```
  ptr = a;
  ```



a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[8]
a[9]

ポインタ変数に対する加減算で相対位置を指定することができる！

例6

pointer2.c

```
#include <stdio.h>

int main (void) {
  int a[5] = {1, 2, 4, 8, 16};
  int i = 0;
  int *ptr = a;

  while (i < 5) {
   printf("a[%d]: %d¥n", i, a[i]);
   printf("a[%d]: %d¥n", i, *(ptr + i));
   printf("a[%d]: %d¥n", i, ptr[i]);
   i++;
  }

  return 0;
}
```


確認

- （確認10）例6で，ptr + i の値（アドレス値）を表示してみて，繰り返しで幾つずつ大きくなっているか確認しよう
- （確認11）例6で，10行目の*(ptr + i)の()はなぜ必要なのだろうか？
  ()を外して実行してみて確認しよう.
  - ヒント：演算子の優先順位

printfと文字列と先頭アドレス

- printfで文字列を表示する場合

```
char str[] = "Hello World";
printf("%s¥n", str);
```

文字列を出力するための変換指定子

文字列（配列）の最初の文字のアドレス値を与える必要がある

- **配列名**は**先頭アドレス**（先頭要素のアドレス値）を表す
  - char str[10]; と宣言した場合，
    str と &str[0] は同じ意味（先頭要素のアドレス）

確認

- 以下のプログラムを実行してみよう

```
#include <stdio.h>

int main (void) {
  char str[] = "Hello World";
  printf("%s¥n", str);
  return 0;
}
```

- printf内の str を &str[0] に変えても動くことを確認しよう
- さらに &str[1], &str[2] と変えたらどうなるか？

課題

- 以下の要件を満たすプログラムを作成せよ.
  - 標準入力から20文字以内の文字列(半角英数字)をchar型の配列に読み込む
  - 読み込んだ配列の中身(文字列)を標準出力に出力する
  - 読み込んだ配列の要素を，前から順番にポインタptrで走査し，小文字を大文字，大文字を小文字に変更する
  - 変更した結果を標準出力に表示する.

※標準入力からの入力はscanf関数を使ってよい

※標準出力への出力はprintf関数を使ってよい

※大文字小文字の判定や変換に標準関数は使わないこと

締切は6月29日(火) 23:59

ヒント①

- ポインタによる配列の走査
  - ptr に配列の先頭アドレスを代入した後，必要な回数だけ ptr++ することで，配列の要素を順番に参照することができる

```
int array[10];
int i;
int *ptr;

ptr = array;
while (i++ < 10) {
  *ptr = i;
  ptr++;
}
```


ヒント②

- 大文字かどうかの判定
  - ある char 型の変数 c に代入されている文字が大文字である場合，以下の条件式が真になる

```
c >= 'A' && c <= 'Z'
```

下位 4ビット \ 上位 4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NU	DL	SP	0	@	P	`	p				−	タ	ミ		
1	SH	D1	!	1	A	Q	a	q			。	ア	チ	ム		
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		
6	AK	SY	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	HT	EM	)	9	I	Y	I	y			ゥ	ケ	ノ	ル		
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	VT	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	CR	GS	−	=	M	]	m	}			ュ	ス	ヘ	ン		
E	SO	RS	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	US	/	?	O	_	o	DL			ッ	ソ	マ	゜		

ヒント③

- 大文字を小文字に変換する方法
  - ある char 型の変数 c に代入されている文字が大文字のとき，その文字を小文字に変換するには，小文字のASCIIコードと，大文字のASCIIコードの差を足せば良い