Programming II 0x0b

ポインタと配列（2011.06.16）

塩澤秀和 http://vilab.org

関数にポインタを渡す（復習）

- ポインタ引数
  - 関数にポインタ（変数がある番地）を渡す
  - 関数は番地をたどって変数の中身を変更可能
- scanfの「&」
  - scanf は変数のポインタを受け取る関数
  - なぜそうするのだろう？

```
#include <stdio.h>
void read_int(int *ptr);

int main(void)
{
   int num;

   read_int(&num);
   printf("num = %d\n", num);
   return 0;
}

/* ポインタを受け取る関数 */
void read_int(int *ptr)
{
   printf("整数を入力: ");
   scanf("%d", ptr);
}
```

num の番地を渡す

関数の最初に ptr = # と代入される

ポインタ変数

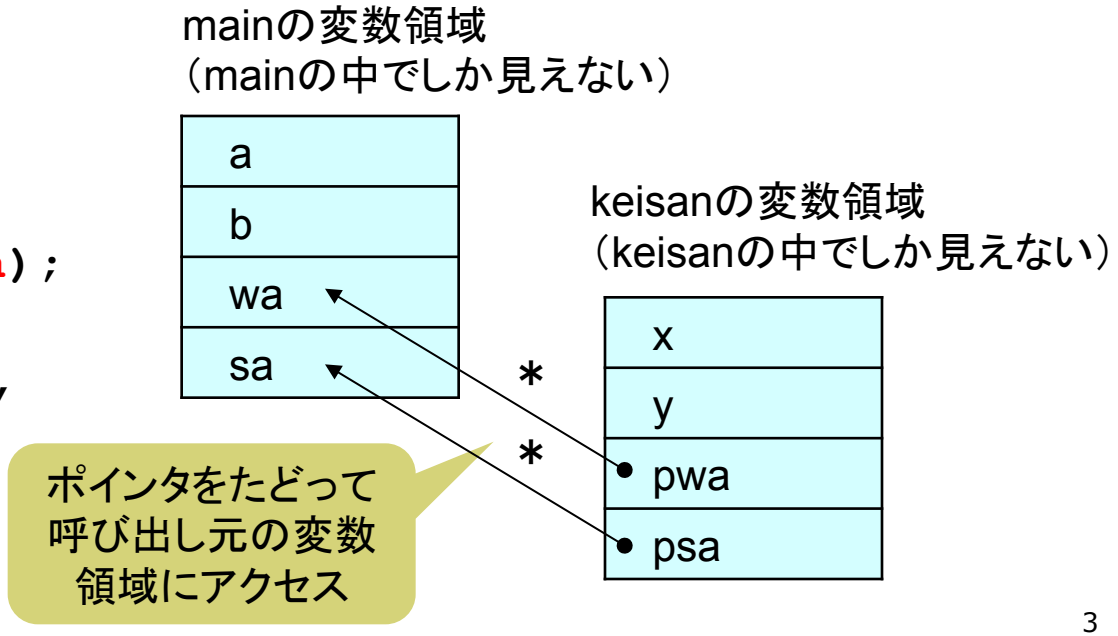
&num と同じ意味

関数から複数の値を戻す（復習）

ここでは分かりやすいように、変数名を変えたが、教科書の「リスト7-12」も理解せよ

```
/* 和と差を一緒に求める関数　*/
/* リスト7-12（p.164）一部変更　*/
#include <stdio.h>

void keisan(int x, int y,
      int *pwa, int *psa);

int main(void)
{
  int a = 2, b = 5;
  int wa, sa;

  keisan(a, b, &wa, &sa);

  printf("和=%d, 差=%d",
         wa, sa);
  return 0;
}
```

```
void keisan(int x, int y,
      int *pwa, int *psa)
{
  *pwa = x + y;
  *psa = x - y;
}
```



配列引数とポインタ（復習）

- 実は、関数に渡しているのは…
  - 配列全体のコピーではなく、先頭要素へのポインタだけ
    ⇒ ポインタを渡しているから、配列の中身を変更できる
  - どちらの形式で書いても意味は同じ（自分で確認せよ）

	配列としての書き方	ポインタとしての書き方
関数呼び出し	`multi2(a);`	`multi2(&a[0]);`
関数定義	`void multi2(int b[])` `{`	`void multi2(int *b)` `{`

- 配列の名前≒ポインタ（次回）
  - array ⇔ &array[0]
  - ポインタに対しても ptr[i] のような書き方ができる

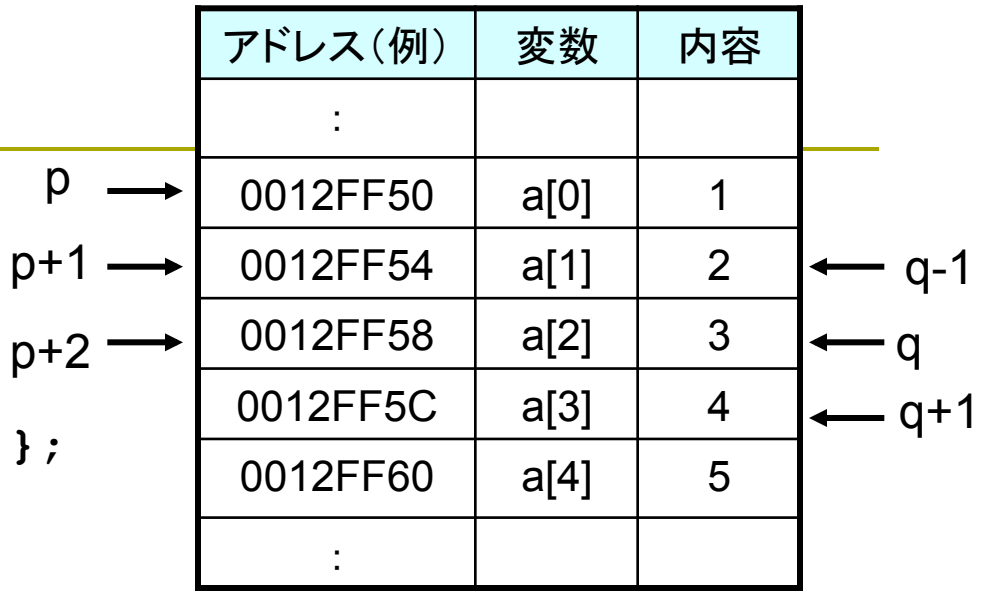
ポインタ演算（p.149）

ポインタと整数の加減算		説明
`ptr + n`		ptr からデータ n 個分進んだ位置を指すポインタ
`ptr - n`		ptr からデータ n 個分戻った位置を指すポインタ
`ptr += n`		ptr = ptr + n
`ptr -= n`		ptr = ptr - n
`ptr++`	`++ptr`	ptr += 1　※後置と前置の違いは通常の変数と同じ
`ptr--`	`--ptr`	ptr -= 1　※後置と前置の違いは通常の変数と同じ

※ ポインタと（素の）アドレス値の違い
　ポインタには指すデータの“型”がある ⇒ データのバイト長を考慮した加減算

ポインタ同士の演算	説明
ポインタの差	`ptr2 - ptr1`（例：&a[4] - &a[2]） 2つのポインタの間に入れられるデータの個数
ポインタの比較	演算子 `< > <= >= == !=` 2つのポインタのアドレス値の大小を比較

ポインタ演算の例




```
#include <stdio.h>

int main(void)
{
  int a[] = { 1, 2, 3, 4, 5 };
  int *p, *q;

  p = &a[0];
  printf("p  = %p番地 → *p    = %d\n", p, *p);
  printf("p+1= %p番地 → *(p+1)= %d\n", p+1, *(p+1));
  printf("p+2= %p番地 → *(p+2)= %d\n", p+2, *(p+2));

  q = p + 2;
  printf("q - p = %d\n", q - p);
  printf("q  = %p番地 → *q    = %d\n", q, *q);
  printf("q+1= %p番地 → *(q+1)= %d\n", q+1, *(q+1));
  printf("q-1= %p番地 → *(q-1)= %d\n", q-1, *(q-1));
  return 0;
}
```

配列とポインタの関係

教科書には「まったく同じ」と書かれているが厳密には例外がある

- 配列の名前≒ポインタ（p.172）
  - 配列名は、その先頭要素を指すポインタに変換される
    ⇒ ptr = array は ptr = &array[0] と同じ意味になる

- 【公式】 x[i] ⇔ *(x + i) （まったく同じ意味）
  - x が配列でもポインタでも成り立つ関係
    配列：a[i] ⇔ *(a + i)　ポインタ：*(p + i) ⇔ p[i]
  - C言語はポインタ中心 ⇒ 実は x[i] は *(x+i) の簡略表記

- ポインタの扱いは慎重に…
  - ポインタ演算は強力なので、バグの原因になりやすい
  - 特に、ptr++ などポインタの値を増減するときは要注意

配列とポインタの記法

```
#include <stdio.h>

int main(void)
{
  int a[5] = { 1,2,3,4,5 };
  int *p;
  int x, i;

  p = a;

  for (i = 0; i < 5; i++){
    printf("a[%d]: %p %p\n",
           i, &a[i], p + i);
  }

  printf("index = ");
  scanf("%d", &i);
```

p = &a[0]
と同じ意味

```
  x = a[i];
  printf("%d\n", x);

  x = *(a + i);
  printf("%d\n", x);

  x = *(p + i);
  printf("%d\n", x);

  x = p[i];
  printf("%d\n", x);

  return 0;
}
```

こう書いても…

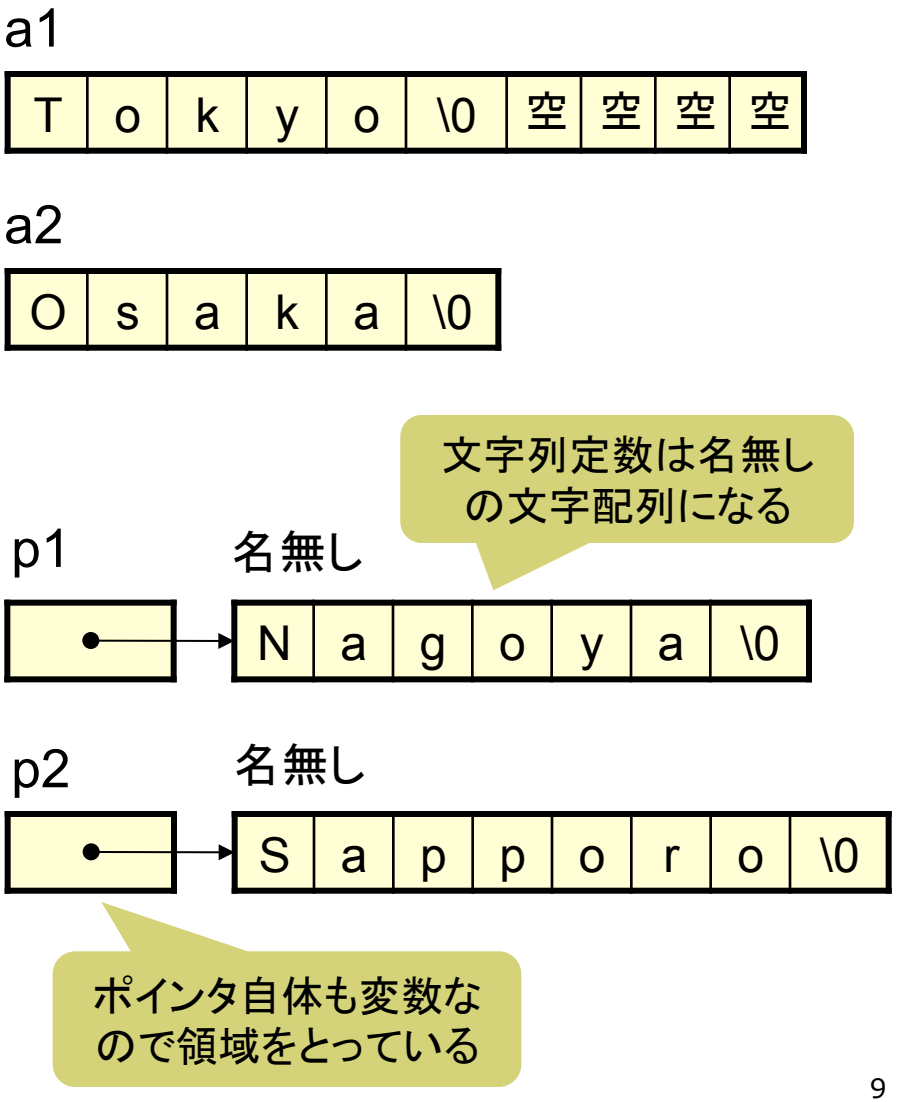
こう書いても…

こう書いても…

こう書いても、同じ

文字列へのポインタ

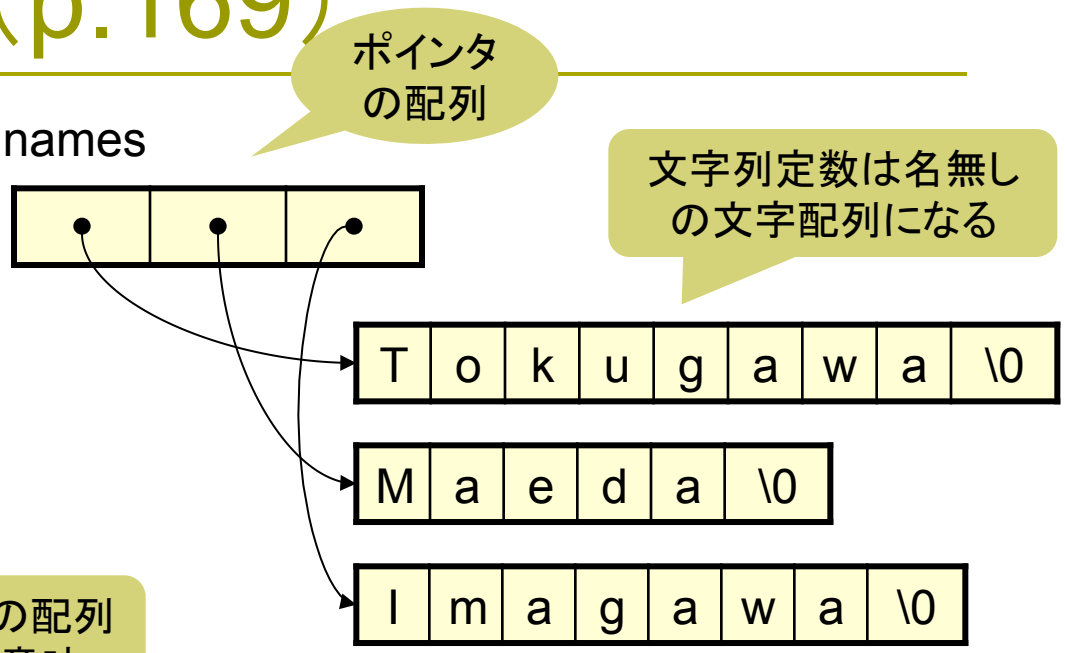
```
#include <stdio.h>

int main(void)
{
   char a1[10] = "Tokyo";
   char a2[] = "Osaka";
   char *p1 = "Nagoya";
   char *p2;

   p2 = "Sapporo";

   printf("a1: %s\n", a1);
   printf("a2: %s\n", a2);
   printf("p1: %s\n", p1);
   printf("p2: %s\n", p2);
   return 0;
}
```



ポインタの配列（p.169）

```
#include <stdio.h>

int main(void)
{
  int i;
  char *names[3] = {
    "Tokugawa",
    "Maeda",
    "Imagawa"
  };

  for (i = 0; i < 3; i++) {
    printf("%s\n", names[i]);
  }
  return 0;
}
```

char * の配列という意味



※ 比較せよ

```
char names[3][10] = {
  "Tokugawa",
  "Maeda",
  "Imagawa"
};
```

2次元配列(復習)

```
#include <stdio.h>

int main(void)
{
  char names[5][10] = {
    "Oda", "Mori", "Takeda",
    "Uesugi", "Shimazu" };
  int i;

  for (i = 0; i < 5; i++) {
    printf("初期値: %s\n", names[i]);
    printf("変更 -> ");
    scanf("%s", names[i]);
  }
  for (i = 0; i < 5; i++) {
    printf("変更後: %s\n", names[i]);
  }
  return 0;
}
```

最大9文字＋ヌル文字

names[0]	O	d	a	\0					
names[1]	M	o	r	i	\0				
names[2]	T	a	k	e	d	a	\0		
names[3]	U	e	s	u	g	i	\0		
names[4]	S	h	i	m	a	z	u	\0	

文字列は配列なので
その配列は文字型の
2次元配列になる

names[i]というのが
1次元配列の名前
（文字列は&不要）

#include <string.h>

- ポインタを使って文字列を処理する関数(p.194)

関数	説明
`strlen(s)`	文字列の長さ(文字数)を返す
`strcpy(s1, s2)`	文字列 s2 を領域 s1 にコピーする (s1 は s2 よりも長い領域がなければならない)
`strcat(s1, s2)`	文字列 s1 の末尾に文字列 s2 を連結する (s1 には十分な空き領域がなければならない)
`strcmp(s1, s2)`	文字列 s1 と s2 を比較する 戻り値は s1<s2 なら負, s1=s2 なら 0, s1>s2 なら正
`strchr(s, c)`	文字列 s から文字 c を探し、最初の位置のポインタを返す(見つからなかった場合は NULL を返す)
`strstr(s1, s2)`	文字列 s1 から文字列 s2 を探し、最初の位置のポインタを返す(見つからなかった場合は NULL を返す)

演習問題

11a. 配列 int a[10] とポインタ int *p を定義し、キーボードから a の内容を読み込んだ後、printf("%d ", *(p + i)); という文を使って a の全要素を表示するプログラムを作成しなさい。

11b. printf(”%s”, &str[i]); とすると str[i] から後ろの部分だけが表示される。標準入力から文字列を読み込み，この機能を使って工夫した表示をするプログラムを作成しなさい。

11c. 整数型のポインタ a および b が指す2つの配列について、先頭から n 個の要素どうしを交換する関数を作成しなさい。

- 関数プロトタイプ　void koukan(int *a, int *b, int n);

11d. 配列 int a[10] にキーボードから値を読み込み、11c で作った koukan を用いて、配列の前半（a[0]～a[4]）と後半（a[5]～a[9]）の内容を交換するプログラムを作成しなさい。