プログラミング１　第５回
ポインタ(2) -- ポインタ演算

- 配列のアドレス
- ポインタ演算（前期教科書P277～）
- ポインタと配列（同上P281）
- 文字列定数と文字列配列（同上P285）

この資料にあるサンプルプログラムは
/home/course/prog1/public_html/2007/HW/lec/sources/
下に置いてありますから、各自自分のディレクトリにコピーして、
コンパイル・実行してみてください

配列のアドレス

- これまで単体の変数でアドレスを考えてきたが、配列の場合はどうであろうか？
- まず、配列の要素アドレスはそれぞれの要素の前に「&」を付加することで知ることが出来る。
- 下の例の中では、配列strのi番目の要素のアドレスは&str[i]である。
- ついでに配列名そのものの値も出力してみて、右の結果からわかるように、配列名は最初の要素のアドレスと一緒であった。実は、これは偶然ではない（これは後ほど述べる）

```
#include <stdio.h>
main()
{
   int i;
   char str[] = "u-aizu";

    for( i = 0 ; i < 6 ; i++)
      printf("%d %p %c＼n",i,
                          &str[i],str[i]);
    printf("＼n%p＼n",str);
}
```

実行結果:

```
s1000001{std1ss1}1: ./a.out
0 efffff9d0 u
1 efffff9d1 -
2 efffff9d2 a
3 efffff9d3 i
4 efffff9d4 z
5 efffff9d5 u

efffff9d0
s1000001{std1ss1}2:
```

アドレスは環境によって違うが、要素のアドレスの差に注目

配列のアドレス

- 前の例において、文字型の配列（本当は、文字列）であったが、他の型の配列も同じように考えられる。
- long型の場合は以下のようになる

```
#include <stdio.h>
main()
{
   int i;
   long array[] = {11,22,33,44};

   for(i = 0 ; i < 4 ; i++)
     printf( "%d %p %d\n",i,&array[i],array[i]);
   printf("\n%p\n",array);
}
```

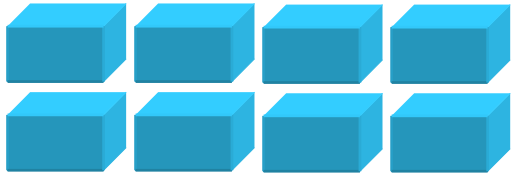
実行結果

```
s1000001{std1ss1}1: ./a.out
0 effff9c8 11
1 effff9cc 22
2 effff9d0 33
3 effff9d4 44

effff9c8
s1000001{std1ss1}2:
```

アドレスは環境によって違うが、要素のアドレスの差に注目

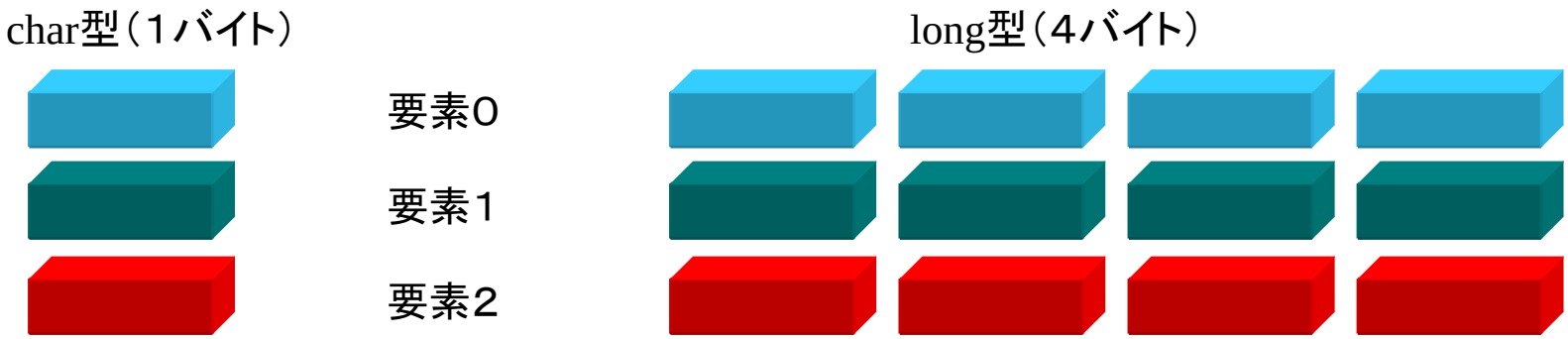
アドレスの飛び方

- 文字型とlong型で分かる通り、配列の各要素は**型の大きさ分だけアドレスが離れている**。
- 各型の大きさを再掲する
- この値は会津大学の標準的な環境での値である

型	大きさ（バイト数）	イメージ
char	1	
short	2	
int,float, ポインタ	4	
double	8	

アドレスの飛び方

- イメージ的には以下のようになる



ポインタ演算

- ポインタ演算（加算・減算）を以下のように取り決める
- **ポインタに1を加える（減じる）とは、ポインタが保持するアドレスに型の大きさ（longなら4）を加える（減じる）ことである。**
- 先ほどのlongの例において、long型のポインタの値が配列の先頭要素（要素0）のアドレス、つまりp = &array[0]だとすると、以下のようになる

ポインタ		アドレス	要素
p	&array[0]	effff9c8	array[0]
p+1	&array[1]	effff9cc	array[1]
p+2	&array[2]	effff9d0	array[2]

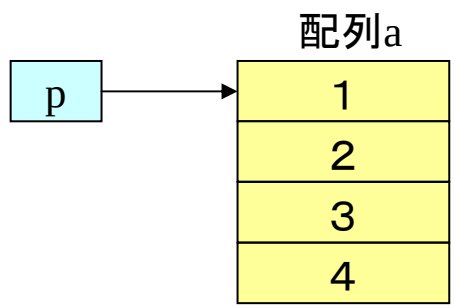
ポインタ演算例

- ポインタ演算を利用すると以下のようなプログラムを書くことが出来る

```
#include <stdio.h>

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

    for(i = 0 ; i < 4 ; i++) printf("%d ",a[i]);
    printf("\n");

    p = &a[0]; /* a の 最初の要素のアドレスを p に代入する */
    for(i = 0 ; i < 4 ; i++) printf("%d ",*(p + i));
    printf("\n");

    for(q = p ; q < p + 4 ; q++) printf("%d ",*q);
    printf("\n");
}
```

実行結果

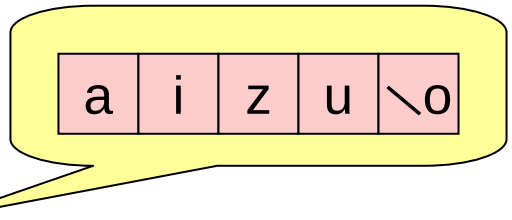
```
s1000001{std1ss1}1: ./a.out
1 2 3 4
1 2 3 4
1 2 3 4
s1000001{std1ss1}2:
```



ポインタ演算例

- 文字列操作の例

```
#include <stdio.h>

main()
{
  int i;
  char str[] = "aizu";
  char *p ,*q ;

  p = &str[0];
  for(i = 0 ; str[i] != '\0' ; i++) printf("%c",str[i]);
  printf("\n");
  for(i = 0 ; *(p + i) != '\0' ; i++) printf("%c",*(p + i));
  printf("\n");
  for(q = p ; *q != '\0' ; q++) printf("%c",*q);
  printf("\n");

  p = q;
  for(i = 1 ; p - i >= &str[0] ; i++) printf("%c",*(p - i));
  printf("\n");
  for(q = p - 1 ; q >= &str[0] ; q--) printf("%c",*q);
  printf("\n");
}
```

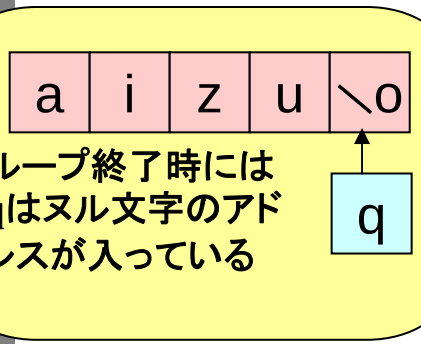


pは&str[0]（配列の先頭）を指す

正順出力

pにヌル文字のアドレスを代入

逆順出力



実行結果

```
s1000001{std1ss1}1: ./a.out
aizu
aizu
aizu
uzia
uzia
s1000001{std1ss1}2:
```

配列名とは

- 文字列の場合、printfなど関数への引数として配列名を渡す。
- 配列名とはいったい何なのだろうか？
- これまでのプログラムの実行結果で分かる通り、**配列名が保持する値は実は最初の要素のアドレスである**。
- つまり配列名とは配列の最初の要素を指すポインタのようなものである（str ⇔ &str[0]）
- 逆に次ページの例のように、ポインタを配列のように使用することも可能である。

配列名とポインタ

```
#include <stdio.h>

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

    p = a; /* つまりこれは p = &a[0] と同じ事 */

    for(i = 0 ; i < 4 ; i++) printf("%d ",a[i]);
    printf("\n");
    for(i = 0 ; i < 4 ; i++) printf("%d ",p[i]);
    printf("\n");
    for(i = 0 ; i < 4 ; i++) printf("%d ",*(a + i));
    printf("\n");
    for(i = 0 ; i < 4 ; i++) printf("%d ",*(p + i));
    printf("\n");
}
```

実行結果

```
s1000001{std1ss1}1: ./a.out
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
s1000001{std1ss1}2:
```

配列風

ポインタ風

配列とポインタの相違点

- 配列とポインタは極めて類似していることが分かった。しかし相違点もある。
- 配列は実際に領域を確保する（正確には 要素数×型の大きさ バイトの領域）のに対して、ポインタはポインタ変数の分（会津大学の通常の環境では4バイト）だけしか確保しない。
- ポインタは適切に初期化しない限り配列の代用にはならない。逆にポインタは適切に初期化すれば配列の代用になると言う事も出来る
- ポインタは同型の変数や、配列をどれを指してもかまわないが、配列名は指定されているメモリ領域しか指すことができず、その値を変更することはできない。つまり、配列名は、ポインタ定数である。

配列とポインタの相違点

- 配列名はアドレス定数であるので、その値（アドレス値）を変更（代入）出来ない。

配列

```
#include <stdio.h>
main()
{
   int a[5]={1,2,3,4,5};
   int i;

   for(i = 0; i < 5; i++) printf("%d¥n", *a++);
}
```

コンパイル
エラー

コンパイル結果
```
   : In function `main':
   : wrong type argument to increment
```

アドレスが指し示すデータの内容をprintfに渡した後で、アドレスをインクリメント(intなので+4)することを意味する

ポインタ

```
#include <stdio.h>
main()
{
  int a[5]={1,2,3,4,5}, *p;
  int i;
  p = a;
  for(i = 0; i < 5; i++) printf("%d¥n", *p++);
}
```

コンパイル、
実行可能！

p[i]のように
書いても良い

配列とポインタの相違点

- ポインタに対して定数初期化は出来ない。
  - 配列は要素の個数分メモリ領域に実際に変数領域が確保される。一方ポインタを配列の代わりに使用しても、実際には領域の確保は行われない。従って下例のようにポインタに対して初期化することは出来ない。

```
#include <stdio.h>
main()
{
  int i,*p={1,2,3,4,5};
  for(i = 0; i < 5; i++) printf("%d¥n", p[i]);
}
```

ポインタ

警告

```
コンパイル結果
  : warning: initialization makes
    pointer from integer  without
    a cast
  : warning: excess elements in
    scalar initializer after `p'
```

実行すると

エラー

```
#include <stdio.h>
main()
{
  int i,a[]={1,2,3,4,5};
  for(i = 0; i < 5; i++) printf("%d¥n", a[i]);
}
```

配列

コンパイル、
実行可能！

配列とポインタの相違点

- 文字列（文字ポインタ）の場合のみポインタに定数初期化が出来る
  - 文字列ポインタの初期化は、定数領域に文字列データが格納され、その アドレスをポインタが指し示すことで行なう。

```
#include <stdio.h>
main()
{
  char a[]="ABC";
  printf("%s¥n",a);
}
```

配列

コンパイル、実行可能！

```
#include <stdio.h>
main()
{
  char *p="ABC";
  printf("%s¥n",p);
}
```

ポインタ

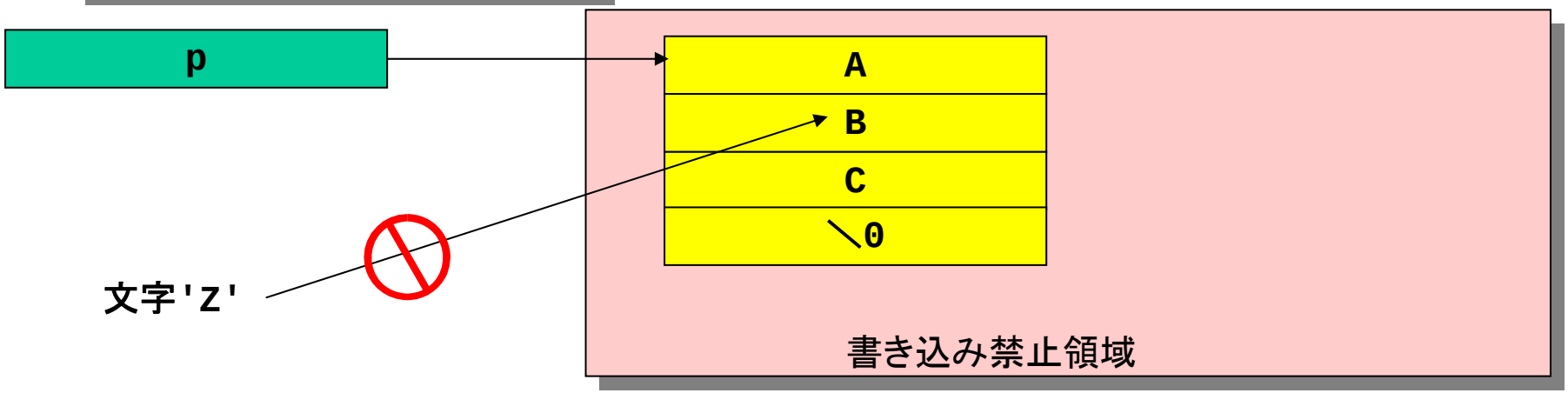
コンパイル、実行可能！

文字列定数の変更

- **文字列定数**はポインタの初期化で宣言する。
- 文字列定数は通常メモリの書き込み禁止領域に領域が取られる
- このため通常の文字列と異なり、文字列定数の変更は出来ない。

```
#include <stdio.h>
main()
{
  char *p="ABC";
  p[1] = 'z';
  printf("%s\n",p);
}
```

実行結果（実行結果はマシンによって異なる）
Segmentation fault



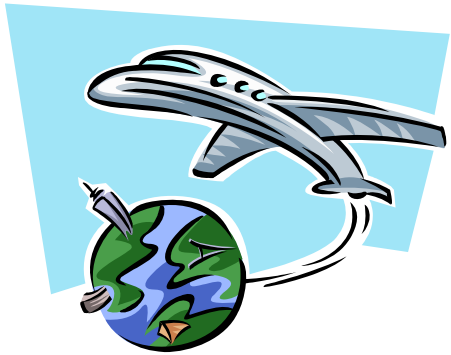
配列とポインタの相違点（まとめ）

- 配列とポインタの違いを表にまとめる
  （aを配列名、pをポインタだとする）

操作	ポインタ		配列	
代入	p = a	○	a = p	×
インクリメント	p++	○	a++	×
加減算	p + 1	○	a + 1	○

文字列定数配列とポインタ配列

- 文字列定数の配列はポインタの配列として定義出来る



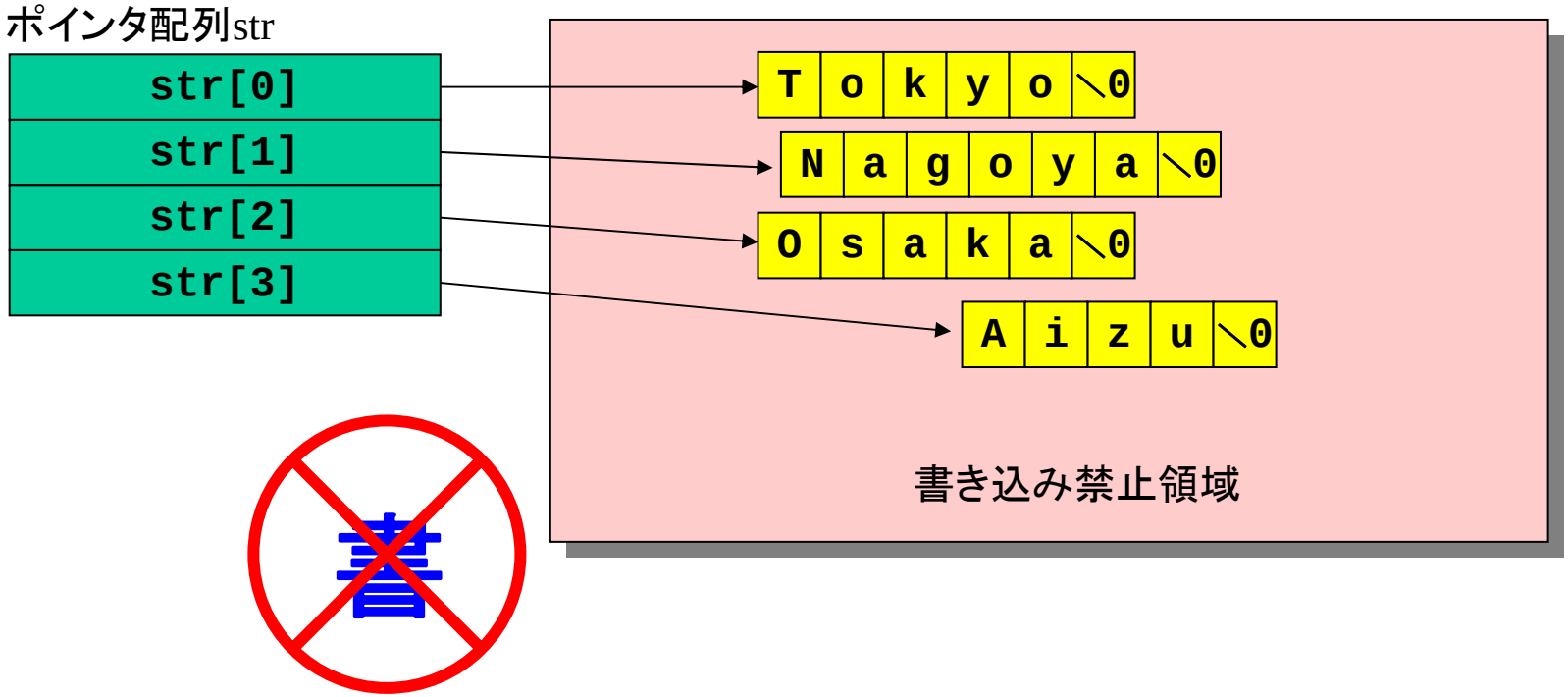
```
#include <stdio.h>
main()
{
  char *str[] = {"Tokyo","Nagoya","Osaka","Aizu"};
  int i;

  for(i = 0 ; i < 4 ; i++){
    printf("%s\n",str[i]);
  }
}
```

実行結果

```
s1000001{std1ss1}1: ./a.out
Tokyo
Nagoya
Osaka
Aizu
s1000001{std1ss1}2:
```

文字列配列とポインタ配列

- この時のポインタと文字列定数の関係は以下の通り
- 文字列としてはstr[2]のように表す。(3番目の文字列を示す。)
- 文字としてはstr[1][2]のように表す。(2番目の文字列中3番目の文字を示す。)



文字列配列を配列で組むと

- この場合は文字列「変数」となるので、自由に代入を行うことが出来る。
- 文字列としてはstr[2]のように表す。（3番目の文字列を示す。）
  str[2]は&str[2][0]と等しい
- 文字としてはstr[1][2]のように表す。（2番目の文字列中3番目の文字を示す。）

```
#include <stdio.h>
main()
{
  char str[4][8] = {"Tokyo","Nagoya","Osaka","Aizu"};
  int i;

  strcpy(str[2],"Sapporo"); /* 文字列の代入可能！ */

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

  for(i = 0 ; str[3][i] != '\0' ; i++){
    printf("%c\n",str[3][i]);
  }
}
```

文字列として表示

文字として縦表示

```
実行結果
s1000001{std1ss1}1: ./a.out
Tokyo
Nagoya
Sapporo
A
i
z
u
s1000001{std1ss1}2:
```

二次元文字配列

- 初期化時の二次元文字配列の状態は以下の通り