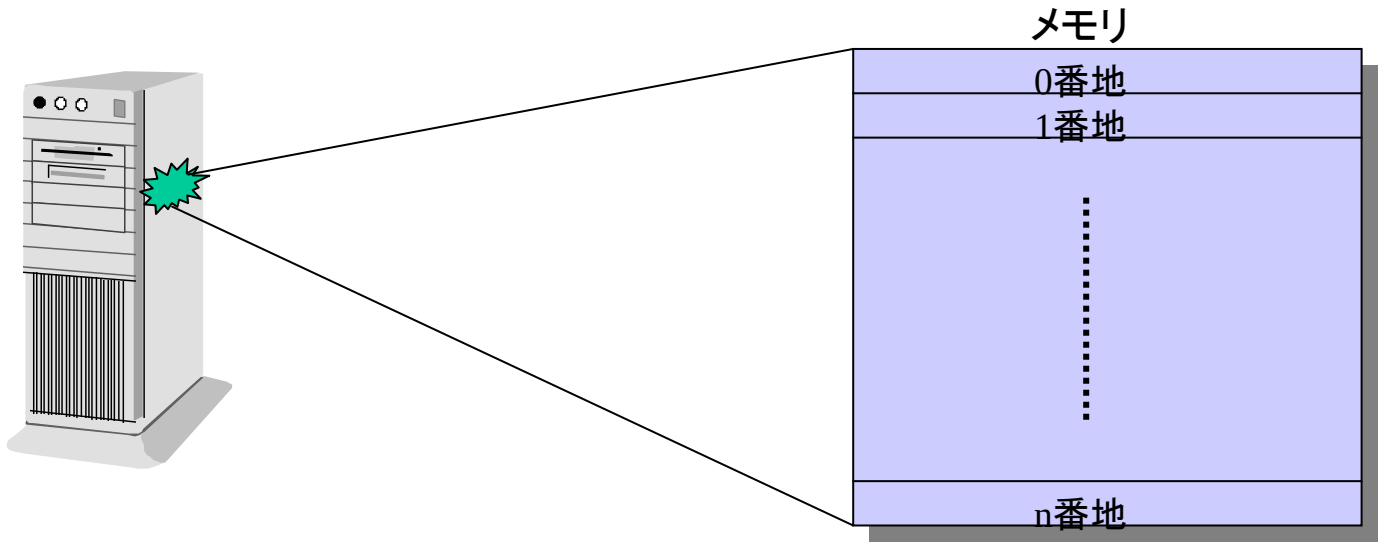
プログラミング１　第４回
ポインタ(1) -- アドレス

- アドレス(前期教科書P270～)
- ポインタ
- 文字列(前期教科書P248～)

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

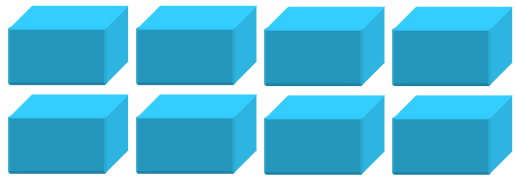
アドレス

- コンピュータはメモリと呼ばれるデータ記憶装置を持つ
- メモリには基本単位（バイトと呼ばれる）毎にアドレスが割り振られている
- アドレスとは「住所」の事であり、データの置かれた「場所」を指し示す
- アドレスは0から順に振られており、同じアドレスが違う場所にも割り当てられることはない



変数の大きさ

- 変数の型によって決まっている変数の大きさは以下の通り
- 厳密にはint型は大きさが決まっていないが、会津大学の環境では4バイトである

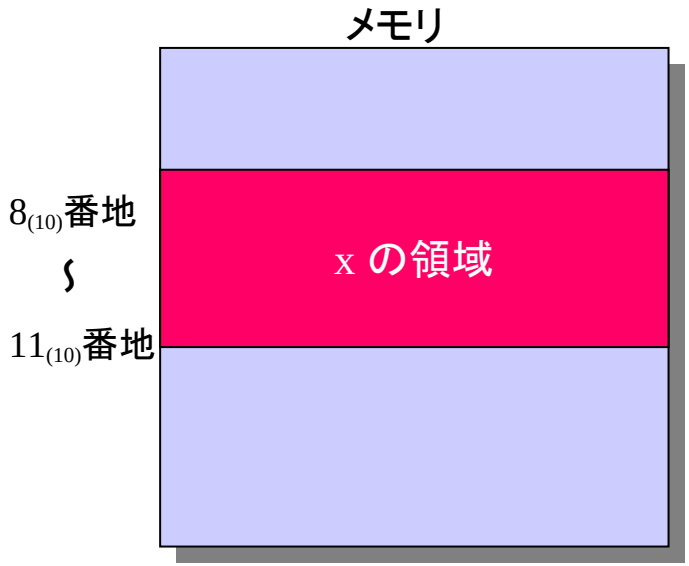
型	バイト数	イメージ
char	1	
short	2	
int,long,float	4	
double	8	

箱一つが1バイト

sizeof 演算子を使用し、
`sizeof double` のようにすると
バイト数が分かる

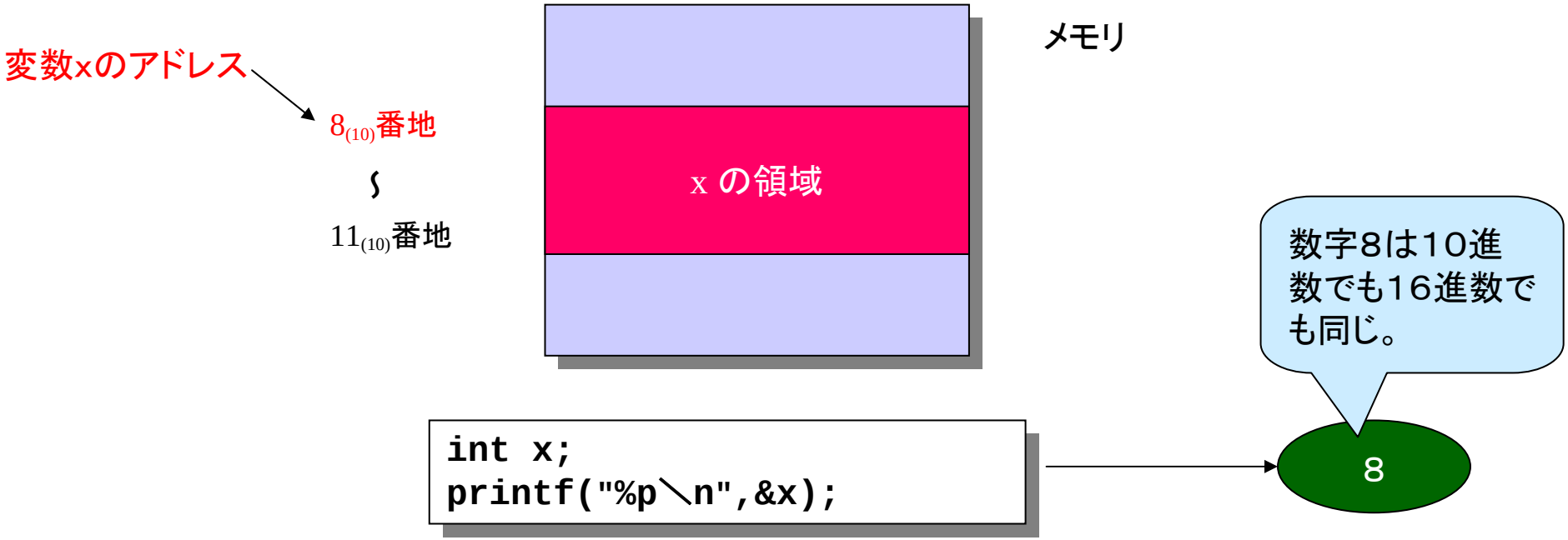
変数の宣言とは？

- 変数の宣言とは、「メモリ内のある場所に、型毎に決められた大きさの場所を名前をつけて確保すること」である。
- 例えば大きさ4バイトのint型変数xを宣言すると、メモリ中に以下のように場所が確保される。（確保される場所のアドレスを仮に$8_{(10)}$番地とした）
- 変数は自動的に適当なアドレスに割り当てられ、ユーザが明に指定することは出来ない。



変数のアドレス

- 前ページの変数xはメモリのアドレス$8_{(10)}$から$11_{(10)}$の領域を占めている。
- この時、先頭の（一番小さい）アドレスを「変数のアドレス」と言うことにする（この場合変数のアドレスは$8_{(10)}$であることになる）
- 変数のアドレスを知るために「＆」演算子がある。
- 例えばこの場合　&x の値は$8_{(10)}$である。
- %pはアドレス専用の書式で16進数表示をする。



ポインタ

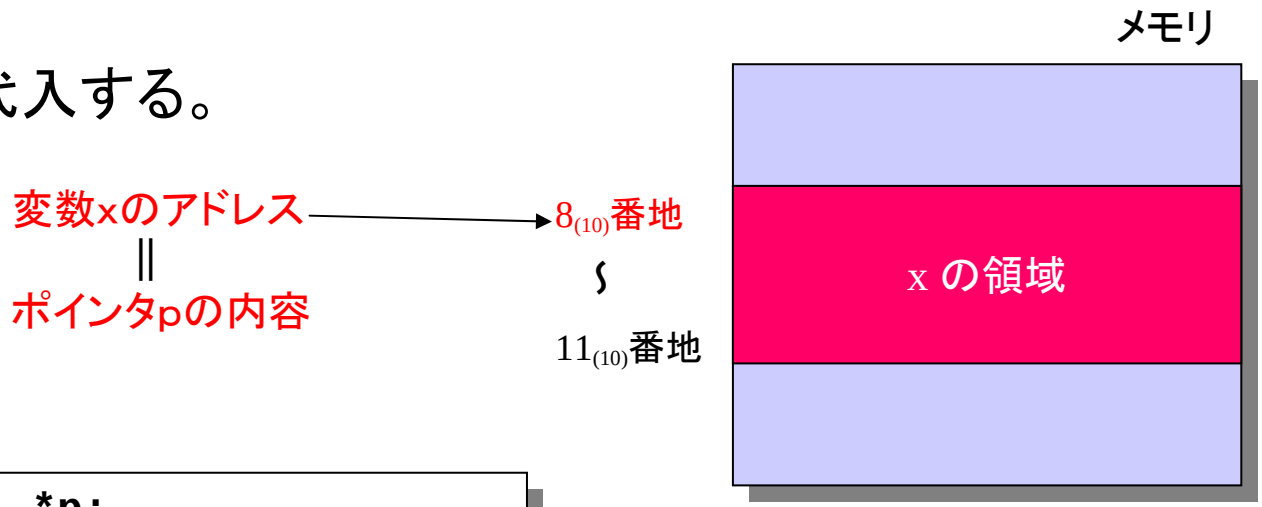
- アドレスを保持することが出来る変数をポインタ(ポインタ変数)と呼ぶ。
- ポインタは各型に「*」(asterisk) を付けて宣言する。
- 例えば、float型のポインタ変数pは、以下のように宣言する。

```
float *p;
```

- ポインタには

```
p = &x;
```

のようにアドレスを代入する。

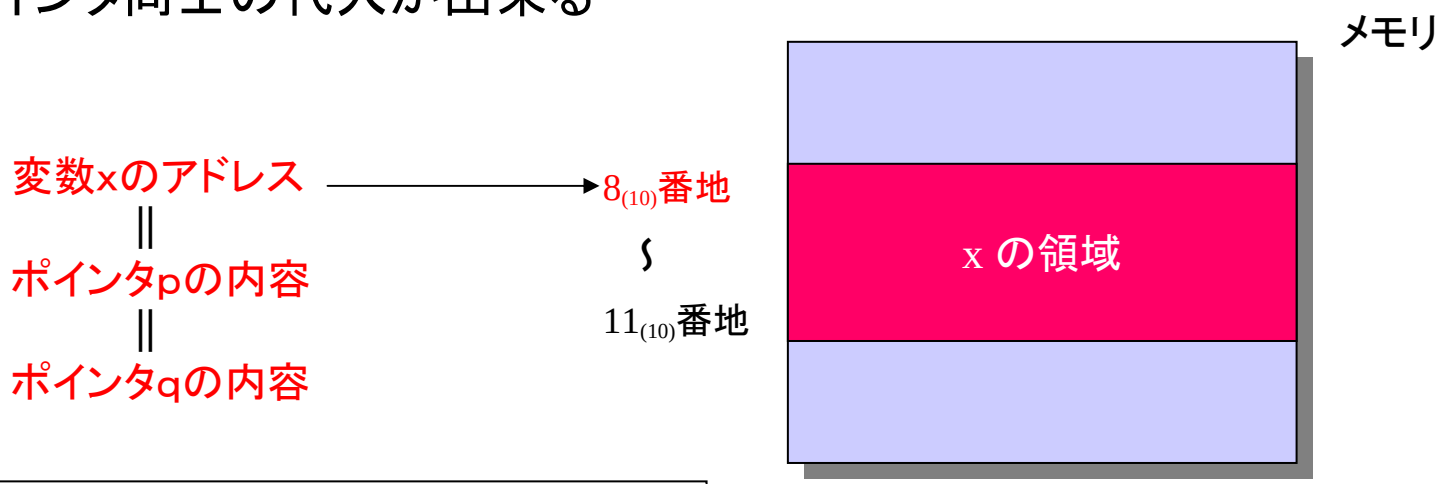



```
float x, *p;
p = &x;
printf("%p¥n",&x);
printf("%p¥n",p);
```

どちらも → 8

ポインタ(続き)

- ポインタには同じ型の変数のアドレスしか代入出来ない。(例えばlong型ポインタはlong型の変数のアドレスを保持する)
- ポインタ同士の代入が出来る

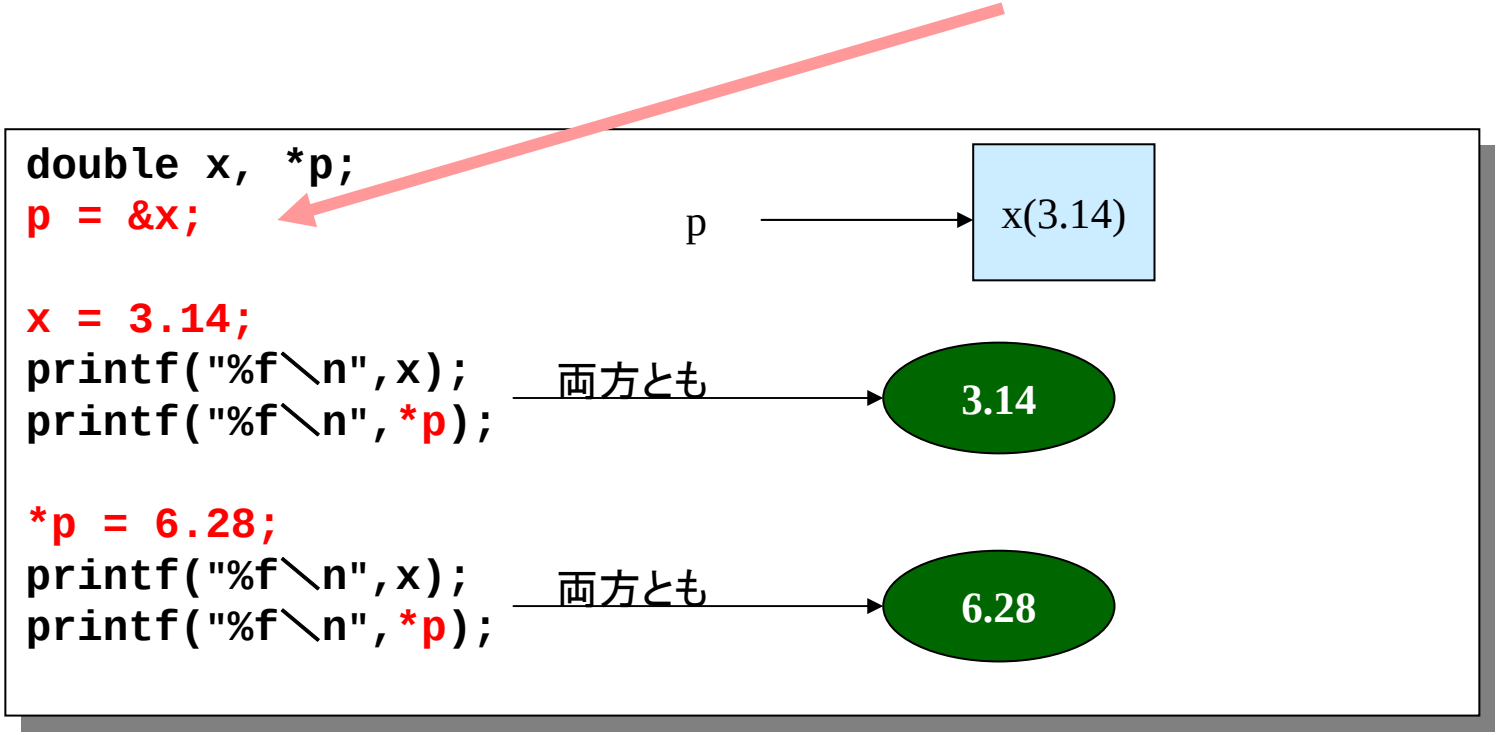



```
long x, *p, *q;
p = &x;
q = p;
printf("%p¥n",&x);
printf("%p¥n",p);
printf("%p¥n",q);
```

どれも → 8

ポインタ(続き)

- ポインタが指し示しているデータの内容を「*」演算子（間接演算子）によって読み・書き出来る。
- 但し、その前に必ずポインタがデータのアドレスを指し示すように何かアドレスが代入されていなければいけない。



ポインタの大きさ

- ポインタも普通の変数と同じように、メモリに「決まった大きさ」を割り当てられる。
- 「決まった大きさ」はOSなどによって異なるが、sizeof演算子で知ることが可能である。(sizeof 変数名、または　sizeof 型)
- ポインタの場合、アドレスを保持するので、文字型のポインタでも、double型のポインタでも大きさは同じである
- 会津大のWSの通常のモードの場合32ビット(4バイト)アドレスなので、sizeofの値は全て4になる。(64ビットモードだと8になる)

```
#include <stdio.h>
main()
{
  char   *p;
  double *q;

  printf("char:%d int:%d double:%d¥n",
         sizeof p, sizeof (int *), sizeof q);
}
```

普通にコンパイル

```
std1ss1{s1000000}1: gcc lec04-2.c
std1ss1{s1000000}2: ./a.out
char:4 int:4 double:4
std1ss1{s1000000}3:

std1ss1{s1000000}4: cc lec04-2.c
-xarch=v9
std1ss1{s1000000}5: ./a.out
char:8 int:8 double:8
std1ss1{s1000000}6:
```

演習室1～4で64ビットモード

一休み

- 今週はポインタの基本中の基本の話しかしないが、ここまでの内容ではポインタの有用性はあまり分からない
- 今週の演習でまずはポインタに慣れて、来週に臨もう
- ここから後は一旦ポインタは置いておいて、ポインタに密接に関係がある「文字列」の話になる。

文字列

- 文字型は1文字を保持するものである
  - 例:`char moji;`
- 複数の文字(文字列)を扱う特別な型はなく、先週習った文字型の配列を使用する。
  - 例:`char string[10];`
- 文字列の終端を示すために文字列の最後には'\0'(アスキーコード:0、「ヌル文字」と呼ぶ)という一文字分のコードを付加する。
- このため、文字列を格納するには、見かけの文字列の長さ+1の大きさの文字配列が必要である。

なぜヌル文字を使うの？

- ヌル文字‘＼0‘は文字列の「おわり」を示している
- 普通の配列の場合は、すべての要素を処理するために、要素の数を関数の引数として与える必要がある
- おわりの記号があれば、要素数がいらなくなるので、処理が便利となる
- 文字列は、文書処理に大量に使われているので、処理の効率化を図るために、ヌル文字が重要な役割を演じている

文字列定数と初期化

- 文字列定数は前後を「"」(ダブルクオート)で囲まれた文字並びである。
(printfの書式等に使用してきた)
  - 例: **"Aizu"**
- 文字列の宣言・初期化時には通常の配列のように1文字づつ要素を与えることが可能である(最後にヌルが必要)
  - **char str[5] = {'A','i','z','u','¥0'};**
- さらに宣言時には文字列定数で初期化することも可能である。
  - **char str[5] = "Aizu";**

文字列の初期化(続き)

- "Aizu Univ"を格納する文字列を宣言するには、9文字+1(ヌル)=10文字 必要であるから、この文字列で初期化するためには最低10個以上の要素を持つ文字配列を宣言する必要がある。
  - `char str[10] = "Aizu Univ";`
  - 又は、以下のように[]内を空白で宣言しても自動的に10個の要素を持つ配列がとられる:
    `char str[] = "Aizu Univ";`

文字列の代入

- 実行時には文字列定数の代入は出来ない。

```
char str[10] = "Tokyo"; ←宣言時はOK！
....
str = "Aizu"; ←実行時はエラーになる！
```

- 実行時の代入はstrcpy文字列関数(string.h)を使用する(Lec04-17,18参照）
  - **strcpy(str,"Aizu");**
  - これは2番目の引数("Aizu")の文字列を1番目の引数(str)にコピーするもの

文字列の入出力

- 文字列の表示(次頁参照)
  - %cでヌル文字になるまで1文字ずつ表示する
    ```
    for(i = 0 ; str[i] != '¥0' ; i++)
        printf("%c",str[i]);
    ```
  - %s(文字列表示用書式)と配列名を使って一度に表示する
    ```
    printf("%s",str);
    ```
- 文字列の入力(次頁参照)
  - %sを使って入力
  - 空白までの文字列＋ヌルを配列に代入してくれる
  - 空白を含む文字列は%sでは入力出来ない
  - この時変数名(配列名)に「&」不要
    ```
    scanf("%s",str);
    ```

文字列使用例

```
#include <stdio.h>
main()
{
  char str1[100], str2[100];
  int i;

  /* %c での読み書き */
  for(i = 0 ;; i++){
    scanf("%c",&str2[i]);
    if(str2[i] == '.'){
      str2[i] = '\0';
      break;
    }
  }
  for(i = 0 ; str2[i] != '\0' ; i++)
    printf("%c",str2[i]);
  printf("\n\n");

  /* %s での読み書き */
  scanf("%s",str1);
  printf("%s\n",str1);
}
```

カウンタ変数だけ更新する無限ループ

この例ではピリオドまでを文字列として入力

データがヌルになるまでループを続ける

空白が来るまでの文字を文字列として読み込む

```
std1ss1{s1000000}1: ./a.out
This is a test.↓
This is a test↓

teststring↓
teststring↓
std1ss1{s1000000}2:
```

文字列に関連するライブラリ関数

- 文字列の代入、比較などはそのままでは行えない。
  このためにライブラリ関数が用意されている。
  - **<string.h> を includeする**
  - **a,bは文字列とする**
  - **strcpy(a,b)** : **bをaにコピーする(ヌル文字も含めて)**
  - **strcmp(a,b)** : **辞書順で文字列a,bを比較する**
    - **a,bが等しいと 0**
    - **a < b(aの方が辞書順で前)で 負**
    - **a > b(bの方が辞書順で前)で 正**

    を返す(ただし値自体は不定)
  - **strlen(a)** : **文字列aの長さ(ヌルを含まない文字数、例えば"Aizu"だと4)を返す。**
  - **その他にもいろいろあるので調べてみると良い**
    (ワークステーションでは、例えば、man strcpyで調べられる)

文字列サンプル

文字列の比較には、<, >,
== などは使えない！

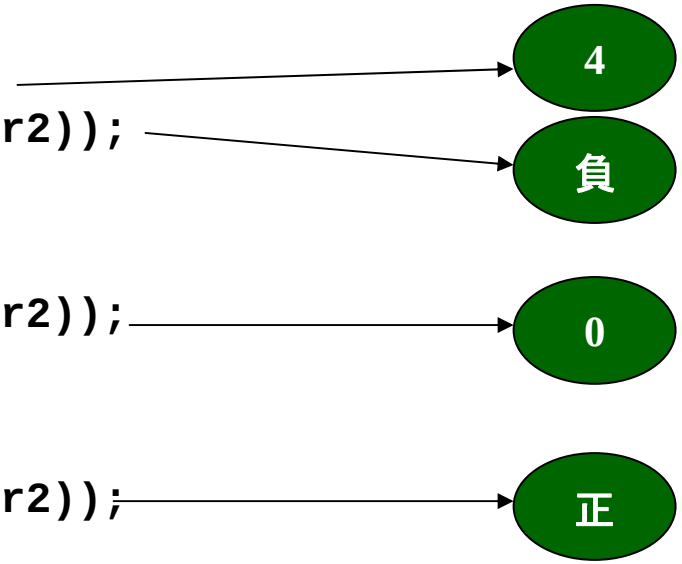
```
#include <stdio.h>
#include <string.h>
main()
{
  char str1[100] = "Aizu", str2[100] = "Aizu univ.";

  printf("strlen %d¥n",strlen(str1));
  printf("strcmp %d¥n",strcmp(str1,str2));

  strcpy(str1,str2);
  printf("strcmp %d¥n",strcmp(str1,str2));

  strcpy(str1,"Aizu university");
  printf("strcmp %d¥n",strcmp(str1,str2));

}
```



文字列サンプル

scanf(”%s”)を使用して、文中の単語の数を数えるプログラム

```
#include <stdio.h>
#include <string.h>
main()
{
  char buf[100];
  int count = 0, acu_leng = 0;

  while(scanf("%s",buf) == 1){
    count++;
    acu_leng += strlen(buf);
  }

  printf("total %d word(s), average length = %f\n",
         count, (float)acu_leng/count);

}
```

実行結果:

```
std1ss1{s1000000}1: cat in1.data
In 1984 Apple Computer introduced the Macintosh
desktop computer with a very "friendly" graphical
user interface. Graphical user interfaces(GUIs) began
to change the complexion of the software industry.
std1ss1{s1000000}2: ./a.out < in1.data
total 28 word(s), average length = 6.250000
std1ss1{s1000000}3: wc in1.data
      4      28     207 in1.data
std1ss1{s1000000}4:
```