プログラミング基礎演習

第6回　ポインタ

担当：矢野史朗 (10-413室)

syano@cc.tuat.ac.jp

お願い

学生掲示板に掲示済みなので対応済みの学生が大半と思いますが
授業中に次の「安否確認システム登録」を行ってください

https://board.cs.tuat.ac.jp/2016/boards/gakusei/body/00011.html

1．前回のおさらい

■文字列のソートについて学んだ

数値も文字列も、アルゴリズムはかわらない

変わらない部分

- 選択ソート：　対象のデータから最小（最大）値を探して、先頭（末尾）と入れ替えを行う
- バブルソート：　隣同士を比較して、大きな（小さな）ものが後ろ（手前）になるよう入れ替える

異なる部分

- 値（＝文字列）の比較に関数が必要
- 値（＝文字列）のコピーにも関数が必要

1．前回のおさらい

strcmp(文字列1, 文字列2)関数

文字列同士を比較し、同じならば0、
文字列1が大きければ正(1など)、小さければ負(-1など)を返す

```
char  StringA[8] = "StrTest";
char  StringB[8] = "StrTest";
if( strcmp(StringA,StringB) == 0)
            printf("SameStr\n);
```

```
SameStr
```

strcpy(コピー先文字列, コピー元文字列)関数

文字列をコピーする関数

```
char  StringA[8] = "StrA2B";
char  StringB[8] = "StrB";
strcpy (StringB,StringA);
printf("%s\n", StringB);
```

```
StrA2B
```

String B = String A
としていないところが大事
理由はポインタの講義で

2. Linux：シェルスクリプトとは

シェル上の一連の操作をひとまとめにしたもの

1. スクリプトファイルを作成（例：下記hoge.sh）

```
#! /bin/bash

ls *.c > filelist.txt &&
cat filelist.txt
```

2. コマンドラインで実行

> bash hoge.sh

2. Linux：シェルスクリプトとは

- 知っておくと，将来的にLinuxを操作し，またプログラミングしていく上でとても役立ちます

  # ただC言語の習得で余裕がない人は，シェルスクリプトは発展的話題程度の気持ちで良いです．

システムに影響するスクリプトや，実験的なスクリプトは
自身の仮想環境等の，周囲に危害を与えない環境で実行してください

2. Linux：シェルスクリプトとは

主に下記を組み合わせて作ります

- 変数
- コマンド置換
- フロー制御(for, if, elif, while, case)
- 関数
- シェルコマンド
- プロセス操作(開始，確認，停止)

今回は，変数，コマンド置換，for文まで

プロセス：動作中のプログラム

変数宣言：直接代入

変数の宣言： **VAR=value**

変数の呼出： **$VAR**

例： VAR=123

echo $VAR

注意：

宣言時に，等号前後にスペースを入れないこと

変数宣言:コマンド置換

コマンド置換: **$(*command*)**

*command*の処理結果を返します

例: VAR=$(ls *.c | grep “_bsort”)

変数宣言は，直接代入と，コマンド置換による代入が頻繁に使われる

for構文

for構文

```
for variable in wordlist
do
        command
done
```

例

```
VAR=$(ls *.c)
for name in $VAR
do
        echo $name
done
```

科学技術計算や,
他ツールとのパイプ処理で
シェルスクリプトが活躍する
場合があります

やや中途半端ですが
今回のLinuxの話は
ここまで

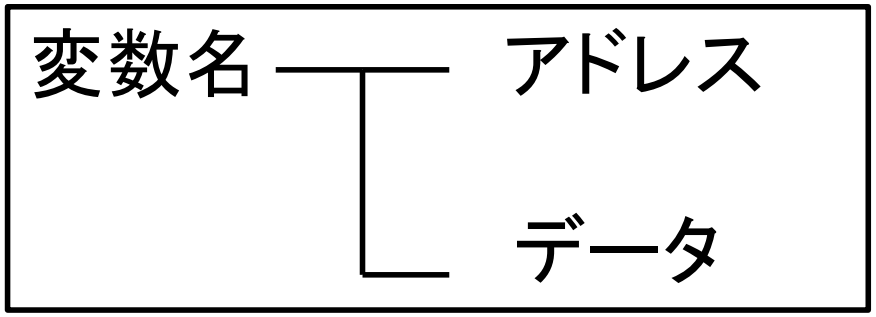
3. ポインタ(プロ序演習の復習)

■メモリの構造と，変数の構造

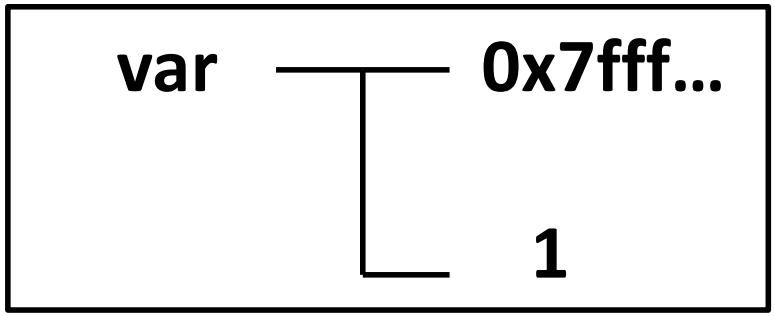
2GByte

アドレス	0	1	2	3	・・・・・	2147483644	2147483645	2147483646	2147483647
データ	00000000	00000000	00000000	00000000	・・・・・	00000000	00000000	00000000	00000000

変数の構造



例 int var=1;



3. ポインタ(プロ序演習の復習)

■アドレスとデータの変換について

&: データ　→ アドレス
*: アドレス → データ

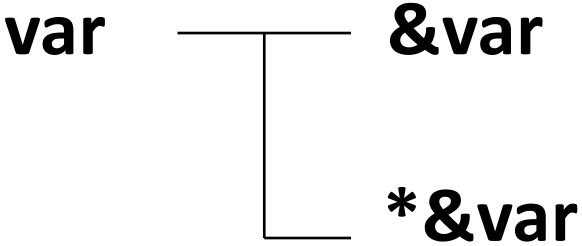
&:アドレス演算子
*:間接演算子

```
int var=1;
printf("      Data: %d¥n",      var);
printf("Address: %p¥n",      &var);
printf("      Data: %d¥n",    *&var);
printf("Address: %p¥n", &*&var);
```

実行

```
Data: 1
Address: 0x7fff…
Data: 1
Address: 0x7fff…
```

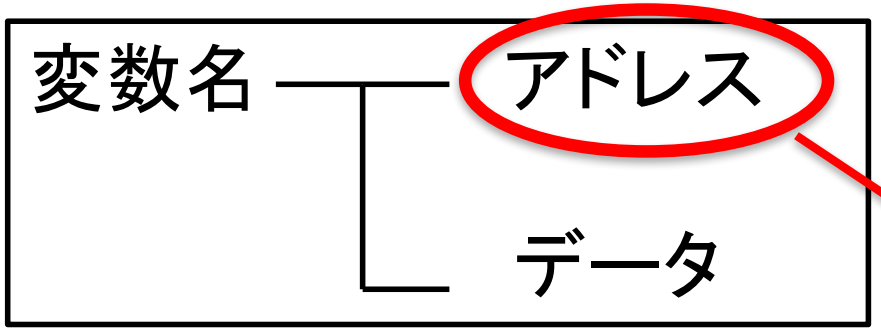
アドレスを呼び出したければ&var
データに戻したければ*&var
(%pはアドレスの16進数表示)



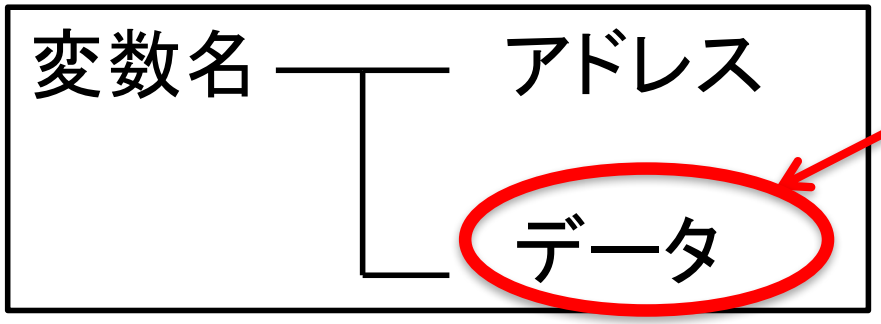
3. ポインタ(プロ序演習の復習)

■変数の構造とポインタ変数の構造

変数の構造



ポインター変数の構造



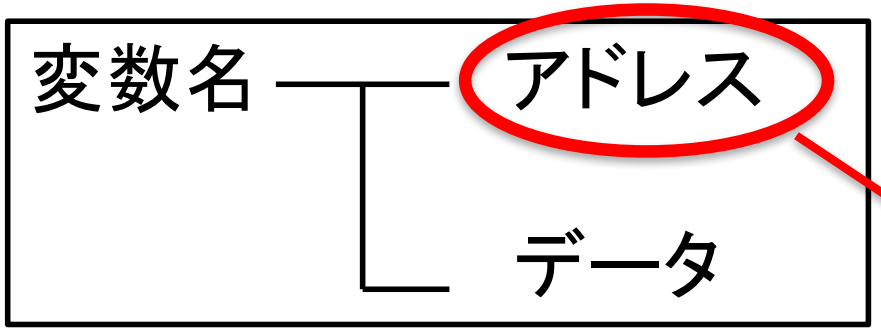
ポインター変数:＝他の変数のアドレスを
データとして格納する変数のこと

・ポインター変数にも勿論アドレスがある

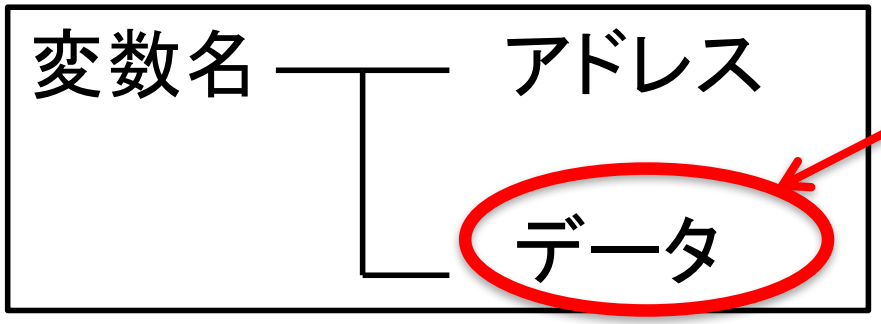
3. ポインタ(プロ序演習の復習)

■変数の構造とポインタ変数の構造

変数の構造



ポインター変数の構造



例

```
int var;
int* p;                    ①

var=1;
p=&var;                    ②

printf("%d", *p);③
```

①pに格納するデータは(int型の)アドレスである，という宣言
②データとしてアドレスを格納
③アドレスをデータに変換

3. ポインタ(プロ序演習の復習)

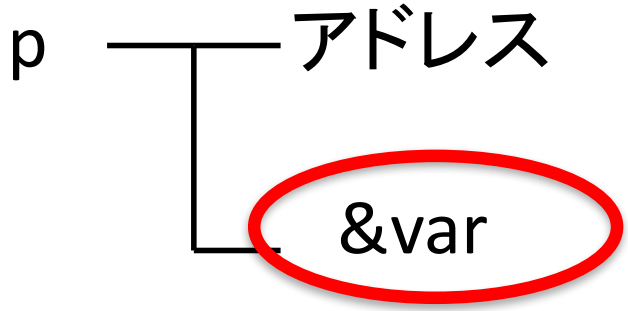
■ポインタ変数，2つの宣言の仕方

宣言の仕方1

```
int var=1;
int* p;          ①

p=&var;          ②
```

①pに格納するデータは(int型の)アドレスです
　という宣言
②アドレスを格納



3. ポインタ(プロ序演習の復習)

■ポインタ変数，2つの宣言の仕方

宣言の仕方2

```
int var=1;
int* p = &var;
```

参照渡し(右記)を扱うにあたって
覚える必要のある表現

「参照渡し(call by reference)」
ポインタ変数を引数とすること

3. ポインタ(プロ序演習の復習)

■ポインタ変数，2つの宣言の仕方

宣言の仕方2

```
int var=1;
int* p = &var;
```

参照渡し(右記)を扱うにあたって
覚える必要のある表現

「参照渡し(call by reference)」
ポインタ変数を引数とすること

```
void func(int* p){…}

int main(){
        int a=1;
        func(&a);
}
```

上記の関数 func は
int* p=&a; のように宣言された
ポインタ変数pを引数にもつ

3. ポインタ(プロ序演習の復習)

■配列とアドレス

```
int var[3]={1, 2, 3};

// varは，var[0]の
// アドレスを指す

printf("%p¥n",&var[0]);
printf("%p¥n",var);
```

配列名:=先頭の値のアドレスを指すポインタ

3. ポインタ(プロ序演習の復習)

■配列とアドレス

```
int var[3]={1, 2, 3};

// varは，var[0]の
// アドレスを指す

printf("%p¥n",&var[0]);
printf("%p¥n",var);
```

配列名:=先頭の値のアドレスを指すポインタ

```
void func2(int* q){…}

int main(){
        int b[3]={1,2,3};
        func2(b);
}
```

上記関数 func2 は
int* q=b; のように宣言された
ポインタ変数qを引数にもつ
(配列名bはアドレスを指すので
この記法が可能)

3. これまでの講義で保留していたこと

■（第5回）文字列操作関数とポインタ

strcmp関数

```
int strcmp(char *s1, char *s2){

    while(*s1 == *s2 && *s1 != '¥0'){
        s1++;
        s2++;
    }
    return *s1 - *s2;
}
```

s1, s2に
配列の先頭アドレスが格納されている

次のアドレスへ

文字コードに基づき，大小を比較

Libc（標準Cライブラリ）によってstrcmp関数の実装は多少異なる

3．これまでの講義で保留していたこと

■（第2回）配列を引数にとる変数

引数に配列をとる場合，変数の局所性とは異なる動作をする

```
void Array_x2(double  x[], int size) {
      int i;
      for(i = 0; i < size; i++)  x[i] *= 2.0;
}

int main(void) {
     double x[10]; int i;
     for(i = 0; i < 10; i++)  x[i] = (double)i;
     Array_x2( x, 10);
     for(i = 0; i < 10; i++)
           printf("x[%d] = %f¥n", i, x[i]);
     return 0;
}
```

```
x[0] = 2.000000
x[1] = 4.000000
x[2] = 6.000000
x[3] = 8.000000
x[4] = 10.00000
x[5] = 12.00000
x[6] = 14.00000
x[7] = 16.00000
x[8] = 18.00000
x[9] = 20.00000
```

3．これまでの講義で保留していたこと

■（第2回）配列を引数にとる変数

double＊ x と全く同じ意味
double型配列の，先頭要素のアドレスを受け取るという意味

```
void Array_x2(double  x[], int size) {
      int i;
      for(i = 0; i < size; i++)  x[i] *= 2.0;
}

int main(void) {
     double x[10]; int i;
     for(i = 0; i < 10; i++)  x[i] = (double)i;
     Array_x2( x, 10);
     for(i = 0; i < 10; i+
           pri
     return 0
}
```

```
x[0] = 2.000000
x[1] = 4.000000
x[2] = 6.000000
x[3] = 8.000000
x[4] = 10.00000
x[5] = 12.00000
x[6] = 14.00000
x[7] = 16.00000
```

配列名のみを書くことで先頭の要素のアドレスを示していた
→　参照渡しなので、変数の局所性は無視される

4. 本日の課題

■スコープ外の変数を処理する関数の実装

main関数内にローカル変数aとbを宣言し，aとbの値をそれぞれa/bの商と余りに変更する，void型の関数divAndRemainderを実装せよ.
ただし，aとbの値は整数である.

【禁止事項】
✕ main関数内でのaとbへの代入
✕ 式による出力（変数単体で出力しなければならない）
　　例）printf(“%d %d¥n”, a/b, a%b); ← これはダメ

4. 本日の課題

■入力の形式

入力は以下の通りである.

```
a b
```

2個の整数値がスペース区切りで入力される.
ただし, 0≦a≦100000, 1≦b≦100000

■出力の形式

以下の通りに出力すること.

```
a′ b′
```

変数a,bを変更した, 変数a'とb'を, スペース区切りで出力する.
なお, b'の後には改行が入る(テンプレートのprintf部分を参照).

4. 本日の課題

■入力例・出力例

例.

入力:

```
8 2
```

出力:

```
4 0
```

入力では，変数a,bの整数型の値がスペース区切りで入力されている.
出力では，値がそれぞれ8/2と8%2に更新された変数a,bの値が出力されている.
なお，出力の0の後には改行が入っている（テンプレートのprintf部分を参照）.

4. プログラムの作成およびコンパイル

■プログラムの作成（第6回講義演習課題）

- ソースコードのファイル名は次の通りにする（すべて小文字）
  （EDENアカウント）_pfunc.c
- 提出用ディレクトリ：
  /home/syano/prokiso/
- コマンドでコピーを行う場合：
  cp （EDENアカウント）_pfunc.c /home/syano/prokiso/

- コピーする前にアクセス権を変更しておくこと

[sxx@xx~/prokiso] chmod 755 （EDENアカウント）_pfunc.c

- 提出期限： 11月21日（月） AM10：30（講義開始時）

5．STEP UP課題（こちらは提出しないでください）

■多次元配列におけるメモリの割り当て

多次元配列ではどのように割り当てているのか調べてみよう

```
#include<stdio.h>  /* 2次元配列におけるメモリの割り当て状況出力プログラム */
#define X 15
#define Y 15
int main(void) {
        int i, j;
        int I[X][Y];

        for(i=0; i<X; i++){
            for(j=0; j<Y; j++){
                I[i][j]=i*X+j;
                printf("I[%d][%d] address: %d, value: %d¥n", i, j, &I[i][j], I[i][j]);
            }
        }
        return 0;
}
```

6. 次回について

第7回（11月21日）
「ポインタと動的メモリ確保」

（再掲）
授業中に次の「安否確認システム登録」を行ってください.
https://board.cs.tuat.ac.jp/2016/boards/gakusei/body/00011.html