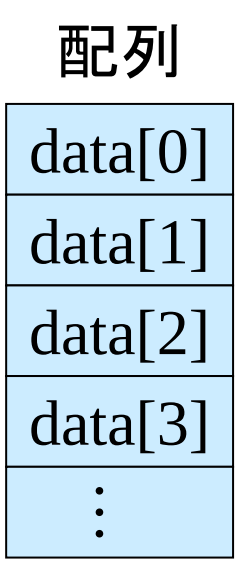
プログラミング１　第８回
構造体(1)・基礎

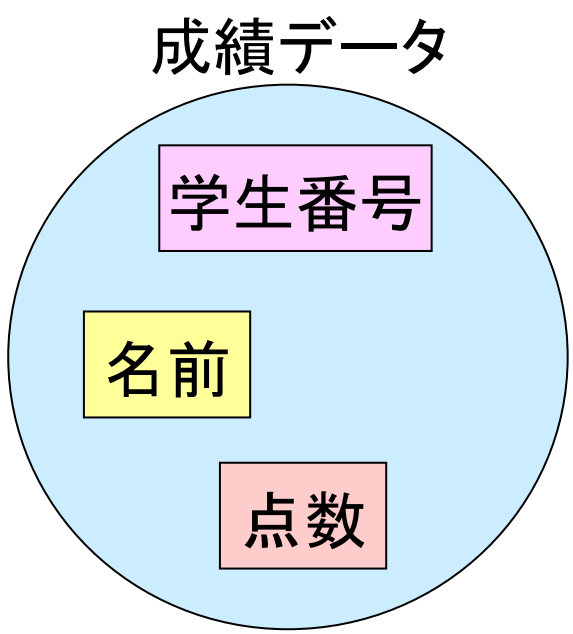
- 構造体の概念・利点
- 構造体の宣言・定義
- 構造体への初期化・代入
- 関数と構造体
- 構造体配列
- 良くあるミス

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

構造体の概念(1)

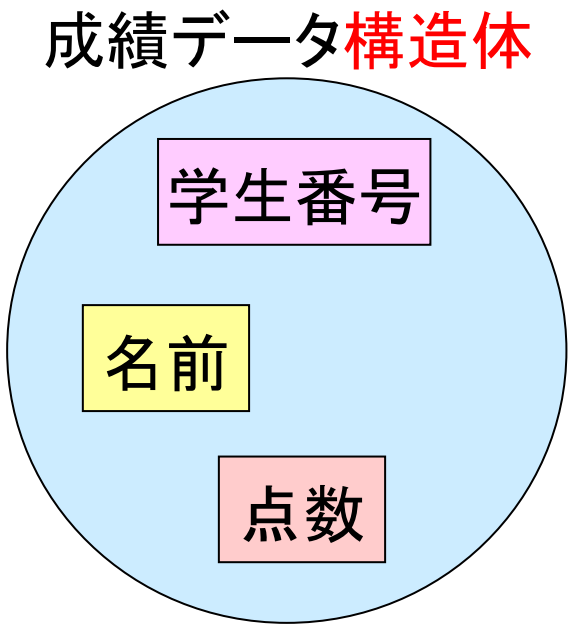
- 関連するデータがある時、同じ型なら、配列にすることでデータをまとめられる。
- 多くの場合、関連するデータは異なった型のデータの集合によって構成される場合が多い。
  - 例えば、学生番号（整数型）、名前（文字型）と点数（実数型又は整数型）など





構造体の概念(2)

- 異なった型のデータ群を、一つのデータ型として宣言する機能を「構造体」と言う。
- つまり構造体とは、異なった型を含むデータをまとめた新しいデータ型なのである。



構造体の利点

- 構造体では 複数の変数を意味のあるカタマリにまとめることが出来、以下の利点がある
  1. 意味が分かりやすいので、プログラムが見やすい
  2. 扱う変数の量を減らせるので、プログラムを簡素化出来る
  3. 「構造体の代入」を利用すれば、メンバーをまとめて扱うことも出来る
- なお、C++やJavaのようなオブジェクト指向言語はこの構造体をもっと拡張して、「オブジェクト」（クラスと呼ぶ）という概念を導入する。
  - 荒っぽい言い方をするなら、オブジェクトは「構造体に関数を含んだようなもの」と言うことが出来る。
    （詳しくは、C++/Javaの本を参照のこと）

構造体の宣言と定義(1)

- 構造体の宣言／定義は次の2段階で行う。
  - 構造体の宣言：
    これは型を作ることに相当する。一度鋳型（いがた）を作れば、それを元に同じ形の物を何個でも作ることが出来る。この型のことを「構造体テンプレート」と呼ぶ。
  - 構造体の定義：
    これは型を元に実際の物を作る作業に相当する。作った物（これを構造体変数と呼ぶ）には通常の変数と同様に一つ一つに他とは違う名前がつけられる。
- 構造体の宣言／定義はstructという、構造体の型を宣言する予約語を使用して定義する。

構造体の宣言と定義(2)

- 構造体の宣言は以下のように行う。

```
struct 構造体タグ名 {メンバー宣言};
```

構造体宣言の例

```
struct roll {
   char name[30];    /* 氏名    */
   long birth;       /* 生年月日*/
   char address[80];/* 住所    */
   int  size[2];     /* 身長体重*/
   int  gender;
              /* 性別 男:0 女:1  */
}; ←セミコロン必要！！
```

- この宣言を構造体テンプレートとも呼ぶ。
- 構造体テンプレートの名前をstructの次に書くが（例の場合はroll）、これを構造体タグと呼ぶ。
- {}の中にこの構造体の構成要素（メンバという）を列挙する。メンバには以下のようなものを含む事が出来る
  - 通常の変数（int,float,charなど）
  - 配列（文字列など）
  - ポインタ（詳しくは第９回、第１０回講義にて）
  - 構造体（構造体の入れ子と言う。詳しくは第９回講義にて）

構造体の宣言と定義(3)

- 構造体の定義は以下のように行う。

```
struct 構造体タグ名 構造体変数名;

（例） struct roll my_data, his_data;
```

- この例では、構造体rollの型を持つ my_data, his_data という2つの構造体変数が実際に定義された。
- 即ち、struct roll は自家製の新種の変数として使える(int, floatなどは、システムが用意してくれたものである)。

typedefを使用する方法

- typedefは既存の型を新たな型を作成するためのもので、以下のように使用する:

```
typedef 既存の型名 新しい型名;

（例） typedef float Height;
```

- これは、プログラムを読みやすくするために有効である。例えば、身長だけを扱うHeight型をプログラムの先頭に以上のように宣言（Height型は実際にはfloat型となる）しておけば、Height型の変数は、次のように定義することが出来る:

```
Height a,b,c;
```

typedefを使用する方法

- typedefを使用してP6,7の構造体宣言・定義を書き直すと右のようになる

```
typedef struct {
        char name[30];
         long birth;
        char address[80];
        int  size[2];
        int  gender;
} Roll;

Roll her_data , your_data;
```

新しいRoll型が宣言された

ヘッダファイル

- タグ付きテンプレートやtypedefを使用する場合は、その部分だけを独立のファイルにして下の例のようにinclude(インクルード)する。「インクルード」とはその場にファイルの内容を挿入することだと思うと良いだろう。
- このファイルはヘッダファイルと呼ばれ、ファイル名は xxx.h のように拡張子（かくちょうし）が「.h」であるファイルである
- ヘッダファイルにはテンプレートやtypedef宣言以外に、関数のプロトタイプ宣言やマクロ定義を書くのが一般的である
- なお、インクルード時に
  - ＜　＞でくくられたファイル名(<stdio.h>など)はまずファイルを決められたディレクトリ（通常 /usr/include/）から探し
  - ”　”("roll.h"など)でくくられたファイルはソースファイルのカレントディレクトリをまず探し、ない場合は＜＞の場合と同じディレクトリを探す

ヘッダファイルの例

```
#include <stdio.h>
#include "roll.h"
main(){
  .......
  struct roll my_data, his_data;
  .......
}
```

roll.h

```
#define LENG 30
struct roll {
  char name[LENG];
  long birth;
  char address[80];
  int  size[2];
  int  gender;
};
```

ヘッダファイルの利点

- 規模の大きいプログラムでの複数のファイル（分割コンパイル）で同じファイルをインクルードすることで同じ構造体やマクロを使用することが出来、統一の取れたプログラムを作成することが可能である。
- 全く別のプログラムでも同じテンプレートを使うことで、プログラムの再利用が可能になる。
- データのやり取りをするプログラム間（例えばデータを書くプログラム、それを読むプログラム）で同じテンプレートを使用することで、データの互換性を保証する。
- テンプレートを変更する際にはプログラム自身を修正することなく、ヘッダファイルを変更して再コンパイルすれば良いので修正が簡単で、間違いが少ない。

構造体の初期化

- 構造体変数の初期化は配列の初期化と同様、列挙して行う

```
struct 構造体タグ 構造体変数名＝{初期値, 初期値, ……};
```

- ここで、＝の後の中括弧{}の中を初期化リストといい、初期化するメンバの初期値をカンマ「,」で区切って表す（下例参照）
- 中括弧は構造体の中に配列がある場合などに階層的に使用出来る。

```
#define LENG 30
struct roll {
  char name[LENG];
  long birth;
  char address[80];
  int  size[2];
  int  gender;
};
```

```
struct roll my_data =
{
  "Jiro Fukushima",
  19910123,
  "Aizuwakamatsu, Fukushima JAPAN",
  {180,65},
  0
};
```

構造体メンバーのアクセス方法

- 構造体の中の要素のことをメンバーと呼ぶ。メンバーのアクセスは、「.」(ピリオド)を用いて以下の形式で行なう。ピリオドは日本語の「の」と読むことができる。

構造体変数名. メンバ名

- 例えば、これまでで宣言した構造体変数my_dataの各メンバーは
  - `my_data.name`
  - `my_data.birth`
  - `my_data.address`
  - `my_data.size[0]` :身長
  - `my_data.size[1]` :体重
  - `my_data.gender`

  などとすることでアクセスできる。

```
strcpy(my_data.name,"Taro Aizu");
my_data.size[0] = 178;
```

文字列の代入は必ずstrcpyで行う

- メンバーには右上の例のようにして値を代入することが出来る。

構造体同士の代入

- 同じ型の構造体変数同士（例えばmy_data , my_copy）で代入が可能である

```
my_copy = my_data;
```

- これは配列を除いてはメンバー同士の代入と同じである。配列は全要素がそれぞれ代入される
- 構造体同士の代入をメンバー同士の代入（通常の配列は要素毎に代入、文字列はstrcpyで代入する必要がある）で書き直すと以下のようになる。
- このように構造体同士の代入はデータの移動が非常に簡単であることが分かる。（但し、構造体変数の代入は同じ型同士の構造体変数に限るので注意が必要）

```
strcpy(my_copy.name,my_data.name);
my_copy.birth = my_data.birth;
strcpy(my_copy.address,my_data.address);
for(i = 0; i < 2; i++)
  my_copy.size[i] = my_data.size[i];
my_copy.gender = my_data.gender;
```

関数での構造体使用(1)

- mainの外で構造体タグを宣言することで(又はヘッダファイルをインクルードすることで)、全ての関数からその構造体タグを利用出来るようになる。
- 更に通常の型のように関数の引数にもすることが出来る。

```
#include <stdio.h>
#include <math.h>
struct xy {
  float  x; /* x座標 */
  float  y; /* y座標 */
};

float dist(struct xy, struct xy);
main()
{
  struct xy p1 = {1.0, 5.0}, p2 = {4.0, 1.0};

  printf("distance between (%3.1f,%3.1f) and (%3.1f,%3.1f) is %f\n",
         p1.x,p1.y,p2.x,p2.y,dist(p1,p2));
}
float dist(struct xy p1, struct xy p2)
{
  return (float)sqrt((p1.x - p2.x) * (p1.x - p2.x) +
                     (p1.y - p2.y) * (p1.y - p2.y));
}
```

```
s1000001{std0ss0}1: ./a.out
distance between (1.0,5.0) and (4.0,1.0) is 5.000000
s1000001{std0ss0}2:
```

関数での構造体使用(2)

- 構造体は関数の戻り値にもすることが出来る。

```
#include <stdio.h>

struct xy {
  float  x; /* x座標 */
  float  y; /* y座標 */
};

struct xy center(struct xy, struct xy);

main()
{
  struct xy p1 = {1.0, 5.0}, p2 = {4.0, 1.0}, pc;

  pc = center(p1,p2);
  printf("center between (%3.1f,%3.1f) and (%3.1f,%3.1f) is (%3.1f,%3.1f)\n",
        p1.x,p1.y,p2.x,p2.y,pc.x,pc.y);
}

struct xy center(struct xy p1, struct xy p2)
{
  struct xy pc;
  pc.x = (p1.x + p2.x)/2.0;
  pc.y = (p1.y + p2.y)/2.0;
  return pc;
}
```

```
s1000001{std0ss0}1: ./a.out
center between (1.0,5.0) and (4.0,1.0) is (2.5,3.0)
s1000001{std0ss0}2:
```

typedefの使用例

- 前頁のプログラムをヘッダファイルとtypedefを使用して書き直すと以下のようになる。

```
#include <stdio.h>
#include "xy.h"

XY center(XY, XY);

main()
{
  XY p1 = {1.0,5.0}, p2 = {4.0,1.0}, pc;

  pc = center(p1,p2);
  printf("center between (%3.1f,%3.1f) and (%3.1f,%3.1f) is (%3.1f,%3.1f)\n",
        p1.x,p1.y,p2.x,p2.y,pc.x,pc.y);
}

XY center(XY p1, XY p2)
{
  XY pc;
  pc.x = (p1.x + p2.x)/2.0;
  pc.y = (p1.y + p2.y)/2.0;
  return pc;
}
```

xy.h

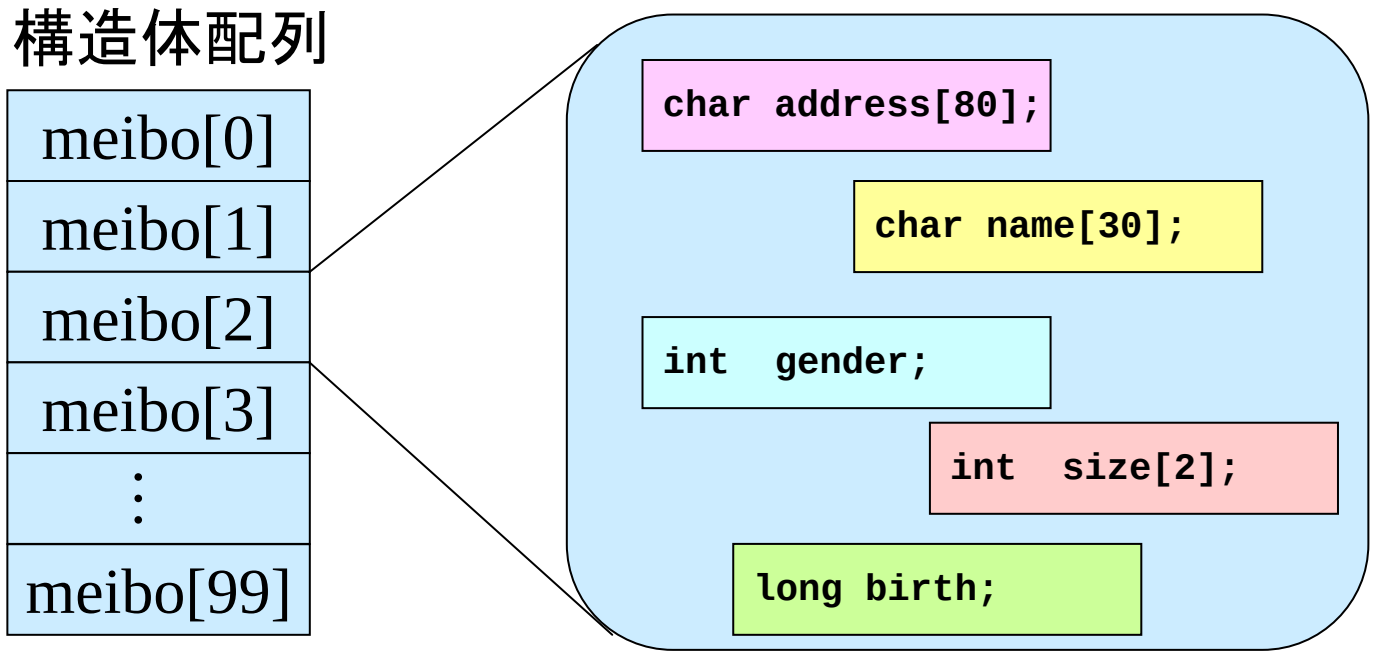
```
typedef struct {
  float  x; /* x座標 */
  float  y; /* y座標 */
} XY;
```

構造体配列(1)

- 構造体変数を配列として宣言することが出来る。
- 例えば構造体テンプレート rollは一人分の名簿の構造体宣言であるので、100人分の名簿を作ろうとするなら、

```
struct roll meibo[100];
```

のようにしてroll型の構造体配列変数を作ってやれば良い。



構造体配列(2)

- 各配列変数内のメンバ参照は

  構造体配列名［添え字］. メンバ名

  で行なう。従って要素1（二番目の要素）への構造体配列の参照は、

```
meibo[1].name
meibo[1].birth
meibo[1].address
meibo[1].size[0] , meibo[1].size[1]
meibo[1].gender
```

  のように記述し、下例のような使い方をする。

```
for(i = 0 ; i < 100 ; i++)
  printf("%d 番目の人の名前 : %s＼n",i,meibo[i].name);
```

- なお、配列を持つ構造体の配列では、上のsizeの例(**meibo[1].size[0]**)のように2重の配列表現が必要になるので、注意が必要である。

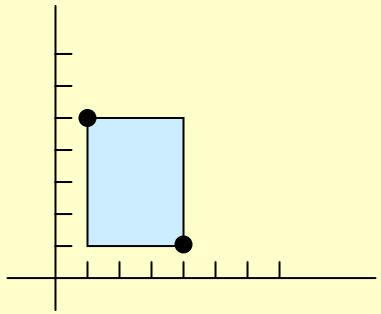
構造体配列の初期化

- 構造体配列変数は配列や構造体と同様に宣言時に初期化出来る。
- 初期化は配列と同様、初期化する要素分構造体初期化データを列挙する。
- この時適宜見やすいように中括弧{}でデータをまとめてやる
- なお、以下のサンプル中、
  - 関数fabsは絶対値(double)を返すmath.hの関数である
  - 長方形は平面上の２点の対角点とそれを通るx軸、y軸に平行な直線で構成されているとする

```
#include <stdio.h>
#include <math.h>
struct xy {
  float  x; /* x座標 */
  float  y; /* y座標 */
};

main()
{
  struct xy rect[2] = {{1.0,5.0},{4.0,1.0}};
  float area;

  area = (float)fabs((rect[0].x - rect[1].x) * (rect[0].y - rect[1].y));
  printf("The area of the rectangle is %f＼n",area);
}
```



```
s1000001{std0ss0}1: ./a.out
The area of the rectangle is 12.000000
s1000001{std0ss0}2:
```

良くあるプログラミングミス(1)

- 構造体を使用した場合の良くあるプログラムの間違いを挙げてみた。
- なお、この節の例は全て構造体タグ**xy**を使用するので、構造体タグの定義は省略した。また**#include**も省略してある

1. 構造体タグの宣言時に、最後の「}」の後のセミコロンを良く忘れる。

```
struct xy {
  float  x;
  float  y;
};
```

これです！

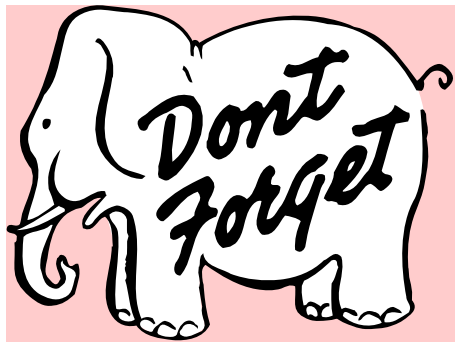
2. メンバーをメンバー名だけでアクセスしてしまう（「構造体名.」を忘れる）

```
main()
{
  struct xy data1 = {1.0,2.0};
  float x = 3.0;
  printf("x: %f  data1.x: %f ¥n",x,data1.x);
}
```

実行結果：

```
x: 3.000000  data1.x: 1.000000
```

（つまり、data1.xとxは全く別物！）



良くあるプログラミングミス(2)

3. 違う型の構造体を代入してはいけない

```
struct xy {定義};
struct ab {定義};
main()
{
  struct xy data1;
  struct ab data2 = {初期化};
  data1 = data2;
}
```

コンパイラのエラーメッセージ:

incompatible types in assignment （代入時の型が不一致）

4. 同じ型でも構造体同士の比較は出来ない

```
main()
{
  struct xy data1 = {初期化}, data2 = {初期化};
  if(data1 == data2)
    printf("等しい！＼n");
}
```

コンパイラのエラーメッセージ:

invalid operands to binary == （＝＝演算子のオペランドが正しくない）

良くあるプログラミングミス(3)

5. 構造体を関数の引数にすると普通の変数と同様「値渡し」になる。
   従って、引数の構造体を変更することは出来ない。(次週アドレス渡しを習う)

```
main()
{
  struct xy data1 = {1.0,2.0}, data2 = {3.0,4.5};
  swap(data1,data2);
  printf("data1:(%3.1f,%3.1f) data2(%3.1f,%3.1f)\n",
          data1.x,data1.y,data2.x,data2.y);
}
void swap(struct xy a, struct xy b)
{
  struct xy tmp;
  tmp = a;
  a = b;
  b = tmp;
  printf("a:(%3.1f,%3.1f) b(%3.1f,%3.1f)\n",a.x,a.y,b.x,b.y);
}
```

実行結果:

```
a:(3.0,4.5) b(1.0,2.0) (関数内では交換されているが)
data1:(1.0,2.0) data2(3.0,4.5) (mainに戻ると交換されていない!)
```