プログラミングI
数理物理，総合理学等向け

2018年11月26日

海谷 治彦

目次

- 期末試験について

- strcpy, strlen 文字列の便利関数 (構造体とは無関係)

- 10章[レ] 12章[明] 構造体について
- 構造体の宣言，変数定義，メンバーアクセス
- 構造体全体の代入 (コピー)
- 関数の引数，返り値

- 前回の演習の解答例
- 本日の演習

期末試験について

- やります.
- 本，紙資料は持ち込み可にします.

- 穴埋め的な，how や what を問う問題は，あまり出しません.
- 何故・・・なのか？的なwhyの問題や，プログラムの読解問題を中心に出す予定です.
- 調べて分かることは調べればよいので，調べるための糸口となることを身に付けてきてください.
  - コレはテスト対策に限らず一般に言えること.

関数とは？

- printf, scanf 等，既に出現している便利な機能がパッケージになっているもの.
- C言語の文法では，if, while, 代入等の基本的な語彙のみが既定されている.
- 便利な機能パッケージである関数は，繰り返し利用することの多い機能を，第三者が予めパッケージ化してくれたものである.

- 該当する関数が存在する場合，関数を使わず，毎回，if や while で処理を書くのは開発効率が悪い.

例 文字列のコピー

- 文字列は文字の配列なので，単純に代入ではコピーできない.
- 次のページにあるように，基本的に，whileやforを使って，1個毎にコピーする必要がある.

- これを毎回やってると開発効率が悪いので，8ページにあるような関数が予め準備されている.
  - 開発効率: 作るのにどれだけ時間がかかるか？
- ただし，include文を追加で記述しないといけない場合が多い.

文字配列の代入 (コピー)

- 配列は通常変数のように丸ごとコピーできない.
- コピーするには要素を一個一個コピー(代入)しないといえない.
- 文字配列の場合，以下の処理を行うが，同じ処理を行う関数 strcpy が用意されている.

```
// Nは適当に定義されているものとする
// src[]は初期化されているものとする
// src から dstにコピーするものとする
char src[N], dst[N];
int i=0;
  while(src[i]) {dst[i]=src[i]; i++;}
  dst[i]=src[i]; // '¥0' のコピーもする
```

strcpy ついでに strlen

- それぞれ文字列に関する操作関数
- #include <string.h> が必要

```
char* strcpy(char* dst, char* src)
```

- srcからdstにコピーする. 方向をすぐに忘れるわ
- 返値は dst と同じ値を返す.

```
size_t strlen(char* s)
```

- 文字列sの長さを返す, '¥0'は含まない長さ.
- sは更新されない, すなわち, 副作用は無い.
- size_t は当面 int と同じと考えてよい.

サンプル

```
// str.c 文字列系関数のサンプル

#include <stdio.h>
#include <string.h>

int main(void){
char buf[101], dest[101];
  // scanfの実行成功した返値は読み込んだ変数の数
  if(scanf("%100s", buf)!=1) return 1;

  // コピーを行う 引数2から引数1へ
  strcpy(dest, buf);
  // 用心のため元をつぶす
  strcpy(buf, "");

  printf("buf=[%s] dest=[%s] %d¥n",
    buf, dest, strlen(dest));

  return 0;
}
```

```
bash-3.1$ cc str.c
bash-3.1$ ./a
sakana
buf=[] dest=[sakana] 6
bash-3.1$
```

関数の標準化

- C言語で利用可能な関数はある程度，標準化されている.
  - POSIXやSUS等
- これによって，異なるOS(Win, Mac, Linux等)でも，同じ関数がある程度，利用可能である.
- printf, scanf, strcpy 等，有名な関数群は，ほぼOS間で動作の差異は無い.

- 尚，関数は自作もできる. (次回)

構造体の必然性

- データ処理では複数のデータを「集まり」にして扱うことが多々あります．
  - 名前，年齢，身長，体重
  - 学籍番号，国語の点数，数学の点数，英語の点数
  - 商品名，価格，メーカー，発売日

  等
- 構造体は，このようなデータの集まりを定義するものです．
- 扱いとしては，intやdouble等と同様，型の一種として扱います．
  - プログラマが定義する型と考えてくれれば結構です．
- 前回の配列は同じ種類のデータの集まりに対して，構造体は異なる種類のデータの集まりといえます．

構造体の操作

- 構造体の宣言
  - intやdoubleと異なり，どんな集まりを新たな型とするか，プログラム内で名言しないといけません．
  - また，名前もつけないといけません．
- 構造体の定義
  - 宣言した構造体に準拠する変数を定義しないといけません．これはintやdouble型の変数と同様です．
- 構造体に参照
  - intやdoubleと異なり複数の値をセットで保持しているので，どの値を参照するか示さねばなりません．
- 構造体の更新
  - 参照と同様．

簡単な例

```
// list10-1a.c List 10-1 改
#include <stdio.h>
#include <string.h>

// 構造体 student の宣言
struct student {
  int id; // 学籍番号的なもの
  char name[20]; // 名前
  int kokugo; // 国語の点数
  int sugaku; // 数学の点数
  int eigo;  // 英語の点数
};

int main(void){
struct student taro;

  // 太郎さんのデータを代入
  taro.id=10;
  strcpy(taro.name, "Yamada");
  taro.kokugo=100; taro.sugaku=85; taro.eigo=60;

  printf("%d %s: %d %d %d¥n", taro.id, taro.name,
    taro.kokugo, taro.sugaku, taro.eigo);

  return 0;
}
```

```
bash-3.1$ cc list10-1a.c
bash-3.1$ ./a
10 Yamada: 100 85 60
bash-3.1$
```

構造体の宣言

コレはキーワードとしてつける.

```
// 構造体 student の宣言
struct student {
    int id; // 学籍番号的なもの
    char name[20]; // 名前
    int kokugo; // 国語の点数
    int sugaku; // 数学の点数
    int eigo;  // 英語の点数
};
```

メンバー(変数)と呼ばれる構造体の中身. 型はばらばらでもよい.

構造体の(変数)定義

- 通常の int doubleの定義と同じ.
- ただし，型の部分を

```
struct 構造体名
```

と書く.

- よって，定義全体は，

```
struct 構造体名 変数名;
```

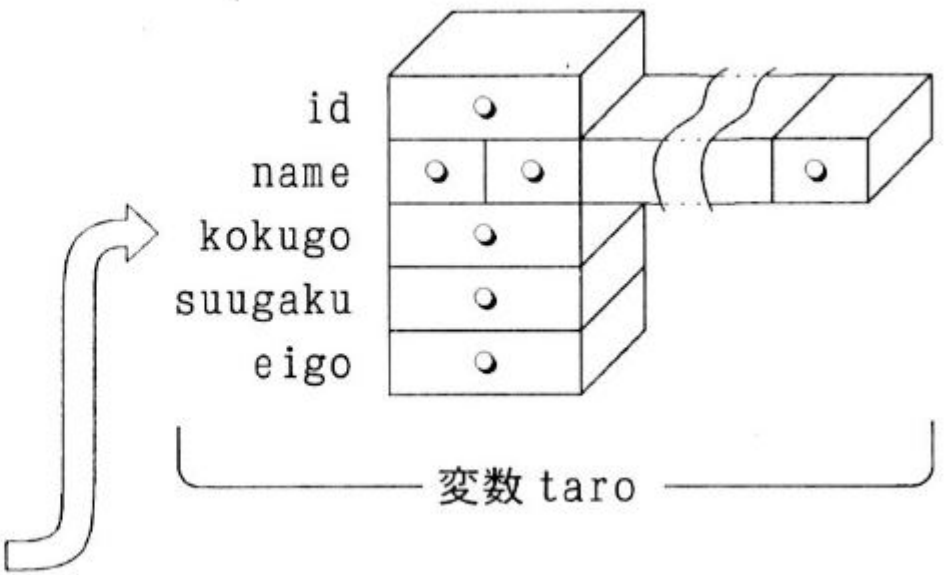
となる.

```
// 構造体の変数定義
struct student taro, jiro;
// 通常型の変数定義
int i, j;
```

宣言と変数定義のイメージ

Cのプログラム

```
struct student { // 宣言
    int id;
    char name[20];
    int kokugo;
    int suugaku;
    int eigo;
};
// 変数の定義
struct student taro;
```



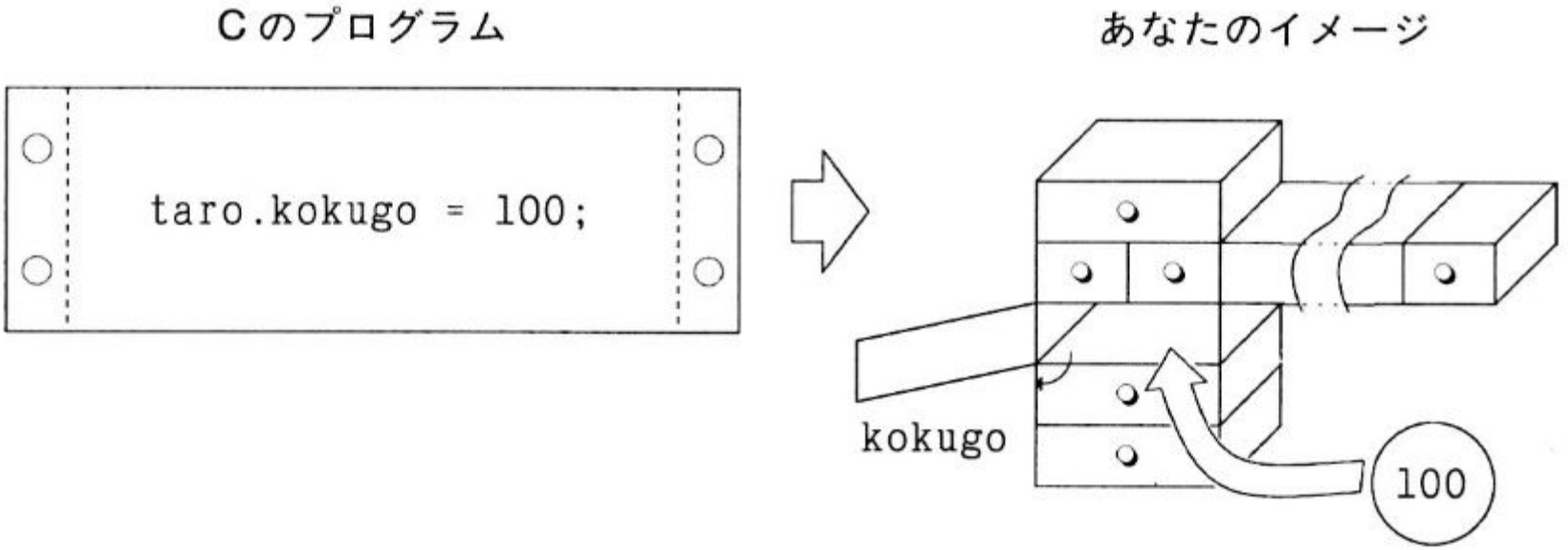
構造体メンバへのアクセス

- メンバに代入や参照するためにアクセスする場合には，. の演算子を用いる. (ドットの演算子)
- 基本，それだけの話.

```
// 構造体 student の宣言
struct student {
  int id; // 学籍番号的なもの
  char name[20]; // 名前
  int kokugo; // 国語の点数
  int sugaku; // 数学の点数
  int eigo;  // 英語の点数
};

// 中略

struct student taro, jiro;
```

```
// 左記の宣言と変数定義に対して，

taro.kokugo=100;

jiro.kokugo=99;

printf("%d %d¥n",
  taro.kokugo,
  jiro.kokugo);
```

メンバアクセスのイメージ



構造体の丸ごと代入

- 構造体型の変数は丸ごと代入できる.
- 中に配列があろうと丸ごと代入される.
- 配列は単独では丸ごと代入できないのに，構造体で括るとできちゃうのは，理不尽だが，そういう文法なので仕方ない.

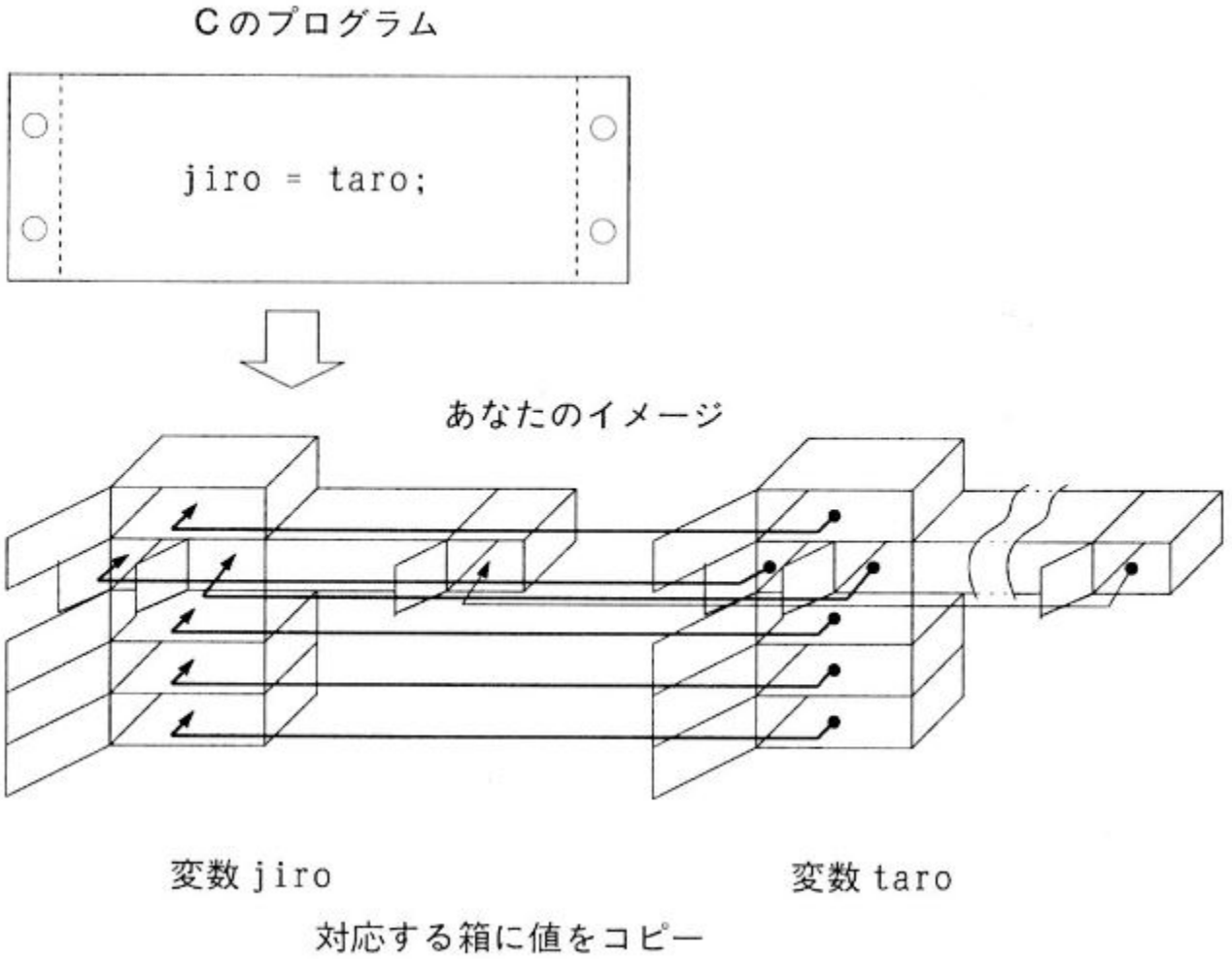
```
// 変数定義
struct student taro, jiro;

// taroのメンバを代入して確定（略）

// jiroはtaroのコピーになる！
// ちゃんと name もコピーされてる.
jiro=taro;
```

サンプル

```
// list10-1b.c List 10-1 改
#include <stdio.h>
#include <string.h>

// 構造体 student の宣言
struct student {
  int id; char name[20]; int kokugo; int sugaku; int eigo;
};

int main(void){
struct student taro, jiro;

  // 太郎さんのデータを代入
  taro.id=10;
  strcpy(taro.name, "Yamada");
  taro.kokugo=100; taro.sugaku=85; taro.eigo=60;

  jiro=taro;

  jiro.id=21; jiro.kokugo=11;

  printf("%d %s: %d %d %d¥n", jiro.id, jiro.name,
    jiro.kokugo, jiro.sugaku, jiro.eigo);

  return 0;
}
```

```
bash-3.1$ cc list10-1b.c
bash-3.1$ ./a
21 Yamada: 11 85 60
bash-3.1$
```

丸ごと代入のイメージ



メンバーの追加

- 構造体のメンバーを追加しても，既存のメンバーにかかわるコードを修正する必要は無い.
- 構造体の変数定義も変更する必要がない.

- 例は list10-2x.c list10-3x.c 等にある.

構造体の初期化

- 構造体も他の変数同様，変数定義の時点で初期を設定することができる.
- 一々代入文を書かなくてもいいから，楽といえば楽.

- サンプル list10-4x.c list10-3y.c 参照

構造体の配列

- intやdoubleと同様に構造体の配列変数を定義することができる.
- イメージとしては，表ですね．(下記，エクセルのように)

```
// 宣言
struct student {
  int id;
  char name[20];
  int ten[3];
};

// 中略

struct student data[3];
```

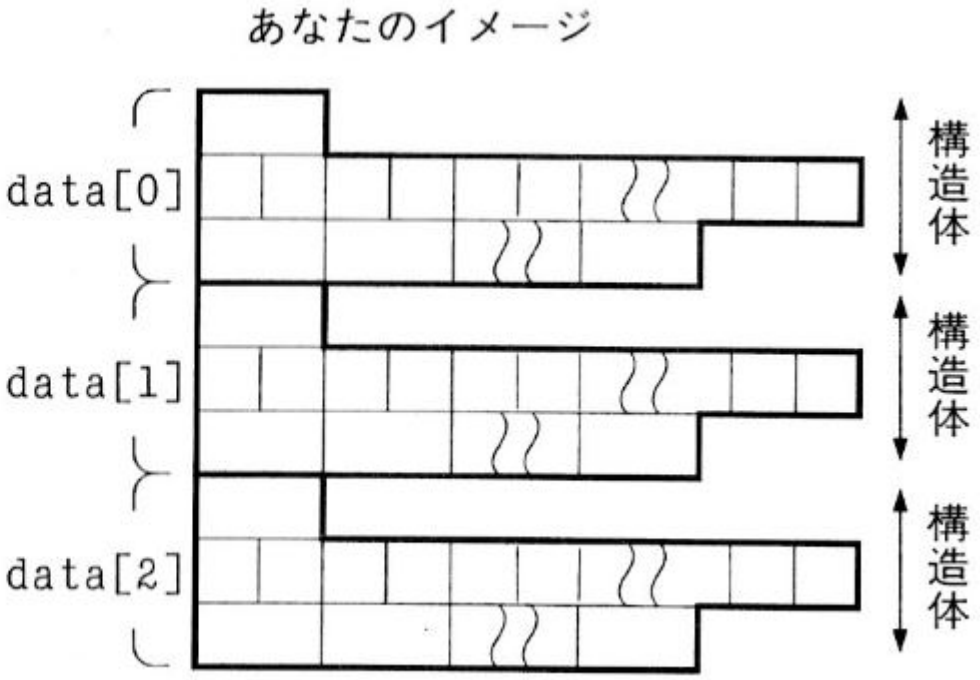
Cだとゼロから順だけど.

	A	B	C	D	E
1	id	name	ten[0]	ten[1]	ten[2]
2	100	Yamada	100	85	60
3	101	Tanaka	99	85	60
4					

教科書のイメージ

Cのプログラム

```
struct student data[3];
```



```c
// list10-3x.c List 10-3 改
#include <stdio.h>

// 構造体 student の宣言
struct student {
  int id; // 学籍番号的なもの
  char name[20]; // 名前
  int ten[3]; // 数科目の点
};

int main(void){
// それぞれ構造体studentの変数を定義
struct student data[3]={
  {1, "Yamada", 81, 82, 83}, {2, "Tanaka", 71, 72, 73},
  {3, "Suzuki", 91, 92, 93}
};
int i=0;
  while(i<3){
  int j=0;
    printf("[%d] %8s:", data[i].id, data[i].name);
    while(j<3){ printf(" %2d", data[i].ten[j]); j++; }
    printf("¥n");
    i++;
  }
  return 0;
}
```

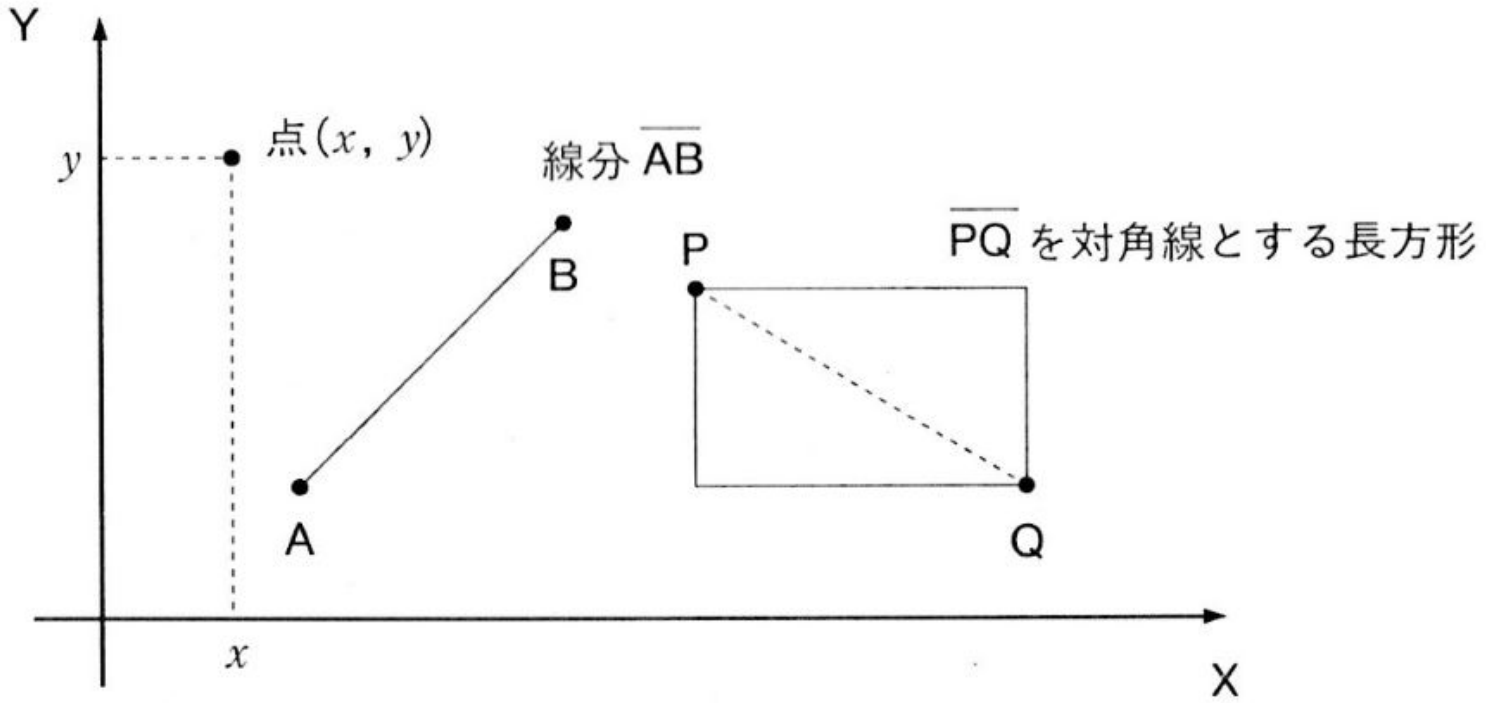
```
bash-3.1$ ./a
[1]   Yamada: 81 82 83
[2]   Tanaka: 71 72 73
[3]   Suzuki: 91 92 93
bash-3.1$
```

構造体メンバとしての構造体

- 構造体は配列以上に普通の型っぽい.
- 複数の異なる構造体の宣言を行うことは当然可能.
- ある構造体の変数が，他の構造体のメンバーになっていることもよくあること.
- 教科書 List 10-6 Fig 10-12では,
  - 点 point は二つの実数からなる構造体.
  - 線 line は二つのpointからなる構造体.
  - 長方形 rectangle も二つの点からなる構造体

  としている.

Fig 10-12のイメージ




```
// l10-6

struct point {
 double x;
 double y;
};
```

```
// l10-6

struct line {
   struct point start;
   struct point end;
};
```

```
// l10-6

struct rectangle {
   struct point p1;
   struct point p2;
};
```

参考
実際の構造体

構造体 task_struct

- include/linux/sched.h の中にある.
- 結構長い，130行くらい.
- 一つのプロセスに関係する情報が全て列挙されいる.

- 主たるものは次のページ

Linux オペーレーティングシステム(OS)の中核となる部分の，構造体の解説です．(ただし，version 2.4 かなり古い)
OSのような実用的システムにおいても，構造体は使われています．まぁ，当たり前ですが.

参考
実際の構造体

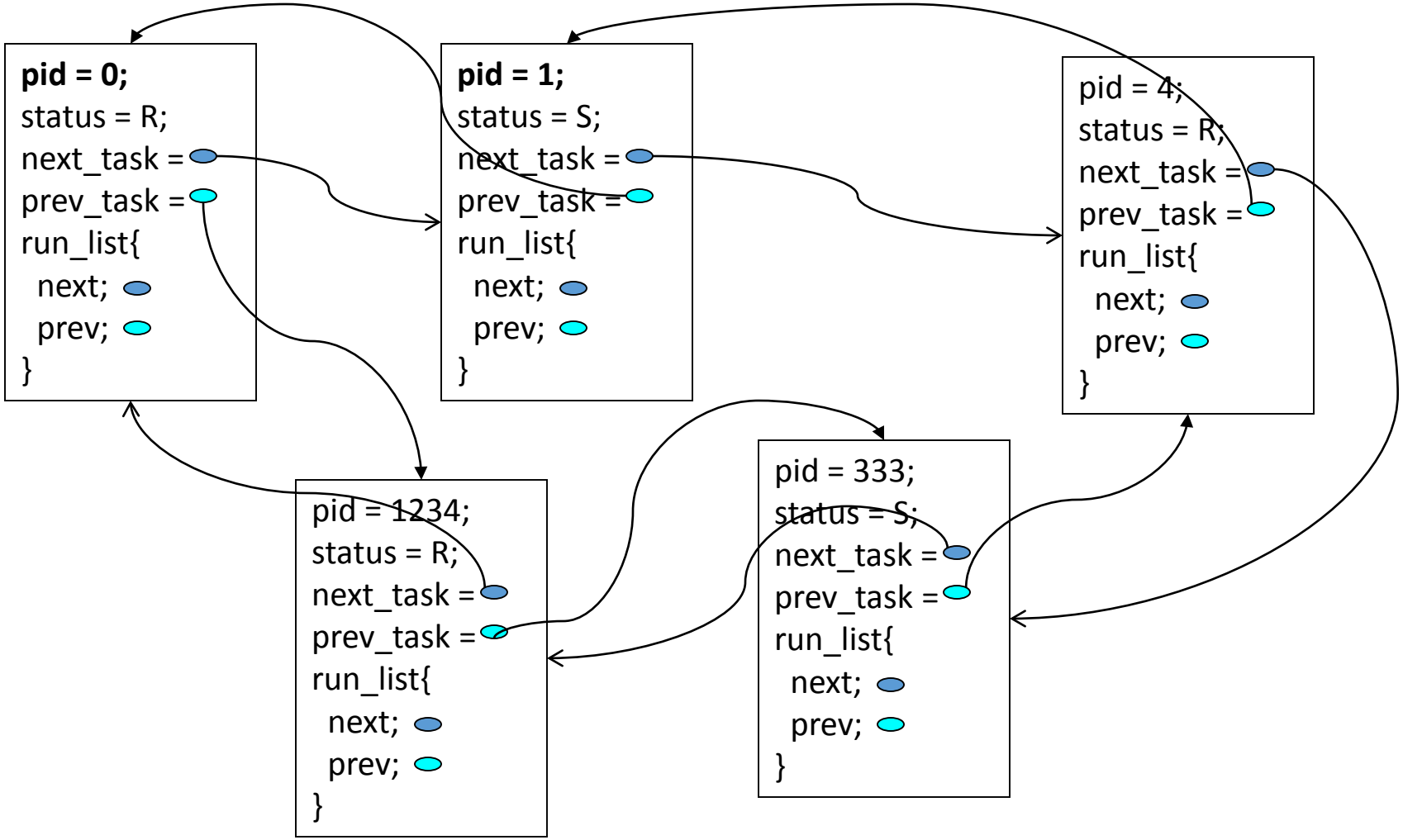
task_structの主たるメンバー

```
struct task_struct { // 無論，抜粋です include/linux/sched.h
  volatile long state;  /* -1 unrunnable, 0 runnable, >0 stopped */
  unsigned long flags;  /* per process flags, defined below */

  volatile long need_resched;
  long counter;
  long nice;

  struct task_struct *next_task, *prev_task;
  struct list_head run_list;

  pid_t pid; // プロセスのID
  struct task_struct *p_opptr, *p_pptr, *p_cptr, *p_ysptr, *p_osptr;

  struct tty_struct *tty; // 対応する端末装置
  struct fs_struct *fs;    // カレントディレクトリ
  struct files_struct *files; // FDへのポインタ
  struct mm_struct *mm; // メモリーリージョンディスクリプタへのポインタ
  struct signal_struct *sig; // 受信シグナル
};
```

```
// include/linux/list.h
struct list_head {
        struct list_head *next, *prev;
};
```

参考
実際の構造体

双方向リストの例



文献5 p.88改

typedef について

- 構造体は「struct 構造体名」で一つの型っぽくふるまう.
- そこで, この, 「struct 構造体名」に型として名前を付ける文法 typedef がある.
- 教科書にもあるが, typedefは構造体専用ではなく, 既存の型に別名を付けるのにも使える.
  - strlen の返値型 size_t もtypedefで定義されている.
- 構造体名と型名との間に何か命名ルールを作っておくと便利.
  - 教科書では struct student は STUDENT としているようだ.
- list10-8x.c 参照.

```c
// List 10-8 改 list10-8y.c
#include <stdio.h>

// 構造体 student の宣言
typedef struct student {
  int id; // 学籍番号的なもの
  char name[20]; // 名前
  int kokugo; int sugaku; int eigo;
} STUDENT; // STUDENT という型を新たに定義した


int main(void){
// それぞれ構造体studentの変数を定義
STUDENT taro={10, "Yamada", 100, 85, 60},
  jiro={11, "Tanaka", 99, 85, 60};

  printf("%d %s, kokugo %3d sugaku %3d eigo %3d, total=%d¥n",
        taro.id, taro.name, taro.kokugo, taro.sugaku, taro.eigo,
        taro.kokugo + taro.sugaku + taro.eigo);
  printf("%d %s, kokugo %3d sugaku %3d eigo %3d, total=%d¥n",
        jiro.id, jiro.name, jiro.kokugo, jiro.sugaku, jiro.eigo,
        jiro.kokugo + jiro.sugaku + jiro.eigo);
  return 0;
}
```

本日の演習 (演習11)

- 以下のような項目を持つ人物の名簿をもとに，体重が80kg以上の人物の名前と身長を列挙するプログラムを作成せよ.
  - 名前，体重，身長，血液型
- テストデータとしては，以下を用いよ.
  - 列挙されるであろう名前は以下である: Okamoto 180, Ohnishi 192
- テストデータは構造体の初期化によって設定してよい.
  - キーボードから入力し，取得する必要は無い.

名前	体重	身長	血液型
Saeki	60	170	AB
Ohnishi	90	192	A
Okamoto	82	180	B
Kato	55	160	B

以下の4バージョン作成せよ

a. w80a.c 前頁の人のデータを記憶するための構造体を定義し，1名分，例えば，Saeki さんの値を持つ構造体型の変数を宣言する.
   - AB型がいることに注意. 1文字では血液型は表現できない.
b. w80b.c 構造体型の変数の配列を用いて，前頁の4人分のデータを変数として宣言する.
c. w80c.c とにかく4人全員の名前と身長を表示.
d. w80d.c 体重が80以上の人の名前と身長のみを表示するように，w80c.c を改造.

本日は以上