J3 Network Exercise

#03 04/28 (Thu) 10:35-12:05

Yasuyuki SAITO
National Institute of Technology, Kisarazu College

基本事項の説明

- 構造体
  - 簡単に言うと、種々のデータを ひとまとまりにしたもの。
  - 構造体の中の各々のデータのことを「メンバ」という。
- 構造体の定義方法と使い方
  - main() の外で、キーワード struct とタグ名を用いて記述する。

```
struct  human {
    double  height;
    double  weight;
    char     name[20];
};

int  main (int  argc,  char  *argv[] ) {
    struct  human    taro;
    struct  human    hanako;
    struct  human    *students;
    ......
```

struct の次にくるキーワードを
タグ名 という。
この例では、human というタグ名
が付けられている。

構造体の定義の最後には ； が
必要なので、要注意!!

基本事項の説明

- 構造体のメンバへのアクセス方法
  - 対象が実体であれば、ドット演算子を使う。ポインタの場合は後述。
    - 前頁の構造体の定義の例でいうと、たとえば以下のように記述できる。
    - 動的に確保したメモリも、[ ] を使えば実体として考える（次ページ）。

```
taro.height  =  172.3;              hanako.height  = 161.4;
taro.weight =  68.9;                hanako.weight  =  53.7;
strcpy(taro.name, “Taro”);      strcpy(hanako.name, “Hanako”);

students = (struct human *) malloc (sizeof(struct human) * 5);
students[0].height = 175.8;
students[0].weight = 72.1;
strcpy(students[0].name,  “Jiro”);
……
students[4].height = 158.8;
students[4].weight = 66.2;
strcpy(students[4].name,  “Goro”);
```

基本事項の説明

- ポインタについて、まずは素朴な変数で考えてみる（構造体ではなく）。

```
double  th;
double  *sh;

th  =  172.3;

sh = (double *) malloc (sizeof(double) * 5);
sh[0] = 175.8;
.....
sh[4] = 158.8;
```

- 上記のように、sh 自体はポインタだが、[ ] で要素を指定すると、普通の変数と同じように実体として考えられる。
- ちなみに、sh[4] は、*(sh + 4) と同義。この加算は、単純な加算ではないことに注意。詳しくはC言語プログラミングの資料 8.3節を参照。
- 式文で用いている変数がポインタかどうか ではなく、実際にアクセスする上で実体（値を扱う）かポインタ（番地を扱う）かを把握することが重要である。

基本事項の説明

- 構造体のメンバへのアクセス方法
  - 構造体変数がポインタであれば、アロー演算子 -> を使う。
    - 以前の構造体の定義の例でいうと、たとえば以下のように記述できる。

```
void set_taro(struct human *t) {
  t->height = 172.3;
  t->weight = 68.9;
  strcpy(t->name, “Taro”);
}

int main(int argc, char *argv[ ] ) {
    struct human  taro;

    set_taro(&taro);

    ……
```

引数を実体とし、ドット演算子を使ってメンバに代入できるが、それは main() の taro とは無関係。
これについては、演習課題で確認。

ただし、もしも t が実体で height がポインタならば、この記述は正しい。とはいえ (*t).height と勘違いしやすいので *(t.height) と書くとよい。

t は番地を扱っているので、
　t->height
と
　(*t).height
は同義。ここで、* よりも . の方が先に評価されるので、括弧が必要。
つまり、もしも括弧がない場合、
　*t.height
は、
　*(t.height)
と同義で、t がポインタなので おかしい。

基本事項の説明

- 構造体の別名定義　当座、このページの内容は理解できなくてもよい。
  - typedef を使って新しい型を名付ける。
    - struct キーワードを使ってもよいし、typedef で付けた型名を使ってもよい。
    - タグ名を省略した場合は、当然ながらタグ名は使えないので、名付けた型名だけが使える。

```
typedef  struct   human {
  double  height;
  double  weight;
  char  name[20];
} hito;

int  main(int  argc,  char  *argv[ ] ) {
   struct  human   taro;
   hito   hanako;
   ......
```

```
typedef  struct  {
   double  height;
   double  weight;
   char   name[20];
} hito;

int  main(int  argc,  char  *argv[] ) {
    hito   taro;
    hito   hanako;
    ......
```

基本事項の説明

- FYI : スペースについて
  - 型名などと変数名との間のスペースは 2 つ空けた方がよいかもしれない（当方は、そうしている）。
    - 構造体の場合は、タグ名と変数名の間。

      struct human taro;
    - プログラミングが高度化するのに伴って、変数宣言が複雑になってくる。
      将来、

      static const struct Foo foo = { 0 };

      const unsigned char *const ptr = puchar;

      のような記述を書くかもしれない。
    - 常に宣言の最後が変数名だけれども、あえてスペースを 1 つ余分に入れて明快にした方がよいと考えている。

      static const struct Foo foo = { 0 };

      const unsigned char *const ptr = puchar;

今回のミッション

- サンプル・プログラムのコンパイル、実行、課題提出
  - 細かい指示は、jes のファイルを参照。
    /home/class/j3/network/expr/src/02_chat/00_readme.txt

- 注意事項（超重要）
  - emacs は授業が終わるまで、終了しないこと。
  - emacs をいちいち複数 起動しないこと。
    - C-x C-f でファイルを呼び出す。存在しないファイル名の場合は、新規ファイルが作成される。ファイルを切り替える場合は、C-x C-b でファイルバッファを開き、C-x o（オー は C- なしの、ただのオー）でカーソルをファイルバッファに移動し、所望のファイルの行で 1 や 2 を押す。
    - その他のファイルバッファ操作やemacs コマンド（copy & paste, search, replace など）は、C言語プログラミングの資料を参照。
  - jes での実習においては、マウス等に触れないこと（CUI なので）。