

第 13 回課題 T13

(新しい型と構造体:教科書第 16 章、2008/7/10(木))

基本問題

T13-1:複素数 (本提出期限 2008/7/10(木)17:40、再提出期限 2008/7/31(木))

提出物 : Makefile、ソースファイル (complex.c)、入力ファイル (complex.in)、出力ファイル (complex.out)

複素数の四則演算 (加算、減算、乗算、除算) を行なうプログラムを作成せよ。ただし、下の実行例を参考に、後ろに示す要求仕様を満すように作成せよ。

complex.in 例

```
1.0 2.0
3.0 4.0
```

実行例

```
b09b0xx@tty:~/T13/1$ ./complex < complex.in
複素数 a=? (空白区切り)
a=( 1.00, 2.00)
複素数 b=? (空白区切り)
b=( 3.00, 4.00)

a+b=( 4.00, 6.00)
a-b=( -2.00, -2.00)
a*b=( -5.00, 10.00)
a/b=( 0.44, 0.08)
b09b0xx@tty:~/T13/1$
```

要求仕様 (T13-1)

全体的な仕様:

- 複素数を表す構造体型として, `Complex` 型を定義して利用せよ. `Complex` 型は以下のよう
に定義してあること.

```
/*構造体テンプレート定義*/  
/*複素数を表す構造体*/  
struct complex  
{  
    double real; /*実部*/  
    double imag; /*虚部*/  
  
};  
  
/*複素数型 Complex の定義*/  
typedef struct complex Complex;
```

- 複素数を標準入力より入力する関数 `scan_complex` および複素数を標準出力へ出力する関
数 `print_complex` を作れ. `scan_complex`, `print_complex` の各仕様は後の説明を参照.
- 複素数の四則演算を行なう関数をそれぞれ作成せよ. 加算を行なう関数 `add_complex`, 減算を
行なう関数 `sub_complex`, 乗算を行なう関数 `mul_complex`, 除算を行なう関数 `div_complex`
の各仕様は後の説明を参照.
- グローバル変数を用いないこと.
- プログラム全体の入力は, 2 つの複素数を標準入力から行なう. 各複素数は, 実部 (`double`
型の値), 虚部 (`double` 型の値) の順に標準入力から入力する.
- プログラム全体の出力は全て標準出力に行なう. 入力された 2 つの複素数と 4 つの演算の
結果すべて (それぞれは複素数) を出力する. 各複素数は (実部, 虚部) の形式で, 実部, 虚
部共にも小数点以下 2 桁まで出力する. 実行例参照.
- `main` 関数中では, `scanf` 文を利用しないこと.
- `main` 関数中では, `printf` 文を用いて複素数型の変数の値の表示を行なわない.
- `main` 関数中では, 複素数の四則演算を行なう関数を呼び出す前に, 入力された複素数を
チェックすること.

入力値によっては, 演算が不可能な場合がある. この場合には, 可能な演算のみ行ない, 不
可能な演算については四則演算の関数を呼び出さずに適切なメッセージを標準出力に出力
せよ.

複素数を標準入力から読み込む関数 `scan_complex` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*  
機能説明 :  
複素数を標準入力から読み込む関数  
  
仮引数 : なし (void)  
  
戻り値 : 標準入力から読み込んだ複素数  
  
副作用 :  
標準入力から 2 つの double 値を読み込み ,  
1 つ目 double 値を戻り値の実部 ,  
2 つ目 double 値を戻り値の虚部とする .  
*/  
Complex scan_complex();
```

- 仮引数はない . すなわち , 仮引数の型は `void` とする .
- 戻り値は , 標準入力より読み込まれた複素数 (`Complex` 型の値) とする . 標準入力より 2 つの実数 (`double` 値) を読みとり , 1 番目の実数を実部 , 2 番目の実数を虚部として持つ複素数を返す .
- 関数 `scan_complex` 中では標準出力への出力は行なってはならない . 例えば , `printf` 文による表示を行ってはならない .

複素数を標準出力へ出力する関数 `print_complex` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*  
機能説明：  
複素数を標準出力に出力する関数  
  
仮引数：z: 標準出力へ出力すべき複素数  
  
戻り値：なし (void)  
  
副作用：  
仮引数で与えられた複素数 z を整形して標準出力へ出力する .  
*/  
void print_complex(Complex z);
```

- 仮引数 `z` は , 標準出力へ出力したい任意の複素数 (`Complex` 型の値) とする .
- 戻り値は無い . すなわち , 戻り値の型は , `void` 型とする .
- 仮引数 `z` で与えられた複素数の実部および虚部を , (実部 , 虚部) の形式で標準出力に出力する副作用を持つ . 実部 , 虚部共に小数点以下 2 桁まで出力する .
- 関数 `print_complex` 中では複素数の値以外の表示を行ってはならない . 特に , “改行” を表示してはならない .
- 関数 `print_complex` 中では標準入力による値の読み込みは行ってはならない . 例えば , `scanf` 文を用いてはならない .

2 つの複素数の和を求める関数 `add_complex` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*  
機能説明 :  
2 の複素数の和を求める関数  
  
仮引数:  
operand1 : 被演算項 1 , 和の左辺の複素数  
operand2 : 被演算項 2 , 和の右辺の複素数  
  
戻り値:  
仮引数で与えられた 2 つの複素数の和 operand1+operand2  
*/  
Complex add_complex(Complex operand1,Complex operand2);
```

- 仮引数は以下の意味を持つ .
 - 仮引数 `operand1` は被演算項であり , 演算の左辺の値 (複素数 , `Complex` 型の値) を表わす `operand1` は , 任意の複素数で良い .
 - 仮引数 `operand2` は被演算項であり , 演算の右辺の値 (複素数 , `Complex` 型の値) を表わす `operand2` は , 任意の複素数で良い .
- 戻り値として , 加算結果 (`operand1+operand2`) の複素数 (`Complex` 型の値) を返す .
- 関数 `add_complex` 中では標準入出力は行なってはならない . 例えば , `scanf` 文や `printf` 文は用いてはならない .

2つの複素数の差を求める関数 `sub_complex` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*  
機能説明 :  
2 の複素数の差を求める関数  
  
仮引数:  
operand1 : 被演算項 1 , 差の左辺の複素数  
operand2 : 被演算項 2 , 差の右辺の複素数  
  
戻り値:  
仮引数で与えられた 2 つの複素数の差 operand1-operand2  
*/  
Complex sub_complex(Complex operand1,Complex operand2);
```

- 仮引数は以下の意味を持つ .
 - 仮引数 `operand1` は被演算項であり , 演算の左辺の値 (複素数 , `Complex` 型の値) を表わす `operand1` は , 任意の複素数で良い .
 - 仮引数 `operand2` は被演算項であり , 演算の右辺の値 (複素数 , `Complex` 型の値) を表わす `operand2` は , 任意の複素数で良い .
- 戻り値として , 減算結果 (`operand1-operand2`) の複素数 (`Complex` 型の値) を返す .
- 関数 `sub_complex` 中では標準入出力は行なってはならない . 例えば , `scanf` 文や `printf` 文は用いてはならない .

2 つの複素数の積を求める関数 `mul_complex` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*  
機能説明 :  
2 の複素数の積を求める関数  
  
仮引数:  
operand1 : 被演算項 1 , 積の左辺の複素数  
operand2 : 被演算項 2 , 積の右辺の複素数  
  
戻り値:  
仮引数で与えられた 2 つの複素数の積 operand1*operand2  
*/  
Complex mul_complex(Complex operand1,Complex operand2);
```

- 仮引数は以下の意味を持つ .
 - 仮引数 `operand1` は被演算項であり , 演算の左辺の値 (複素数 , `Complex` 型の値) を表わす `operand1` は , 任意の複素数で良い .
 - 仮引数 `operand2` は被演算項であり , 演算の右辺の値 (複素数 , `Complex` 型の値) を表わす `operand2` は , 任意の複素数で良い .
- 戻り値として , 乗算結果 (`operand1*operand2`) の複素数 (`Complex` 型の値) を返す .
- 関数 `mul_complex` 中では標準入出力は行なってはならない . 例えば , `scanf` 文や `printf` 文は用いてはならない .

2つの複素数の商を求める関数 `div_complex` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*
機能説明 :
2 の複素数の商を求める関数

仮引数:
operand1 : 被演算項 1 , 商の左辺の複素数
operand2 : 被演算項 2 , 商の右辺の複素数 ((0.0,0.0) 以外の値)

戻り値:
仮引数で与えられた 2 つの複素数の積 operand1*operand2
*/
Complex div_complex(Complex operand1,Complex operand2);
```

- 仮引数は以下の意味を持つ .
 - 仮引数 `operand1` は被演算項であり , 演算の左辺の値 (複素数 , `Complex` 型の値) を表わす `operand1` は , 任意の複素数で良い .
 - 仮引数 `operand2` は被演算項であり , 演算の右辺の値 (複素数 , `Complex` 型の値) を表わす `operand2` に与えられる複素数は , `operand2` は (0.0,0.0) であってはならない . 呼び出された段階で (0.0,0.0) 以外の複素数として良く , この関数内でエラー処理する必要は無い .
- 戻り値として , 除算結果 (`operand1/operand2`) の複素数 (`Complex` 型の値) を返す .
- 関数 `div_complex` 中では標準入出力は行なってはならない . 例えば , `scanf` 文や `printf` 文は用いてはならない .

応用問題

T13-2:ベクトル

(本提出期限 2008/7/17(木)17:40、再提出期限 2008/7/31(木))

提出物: Makefile、ソースファイル (vector.c)、入力ファイル (vector.in)、出力ファイル (vector.out)

3次元ベクトルの内積と外積を求めるプログラムを作成せよ。下の実行例を参考に、後ろに示す要求仕様を満たすように作成せよ。

vector.in 例

```
1.0 2.0 3.0
3.0 2.0 1.0
```

実行例

```
b09b0xx@tyy:~/T13/2$ ./vector < vector.in
v1=?
v1=( 1.00, 2.00, 3.00)
v2=?
v2=( 3.00, 2.00, 1.00)

v1 · v2= 10.00
v1 × v2=( -4.00, 8.00, -4.00)
b09b0xx@tyy:~/T13/2$
```

要求仕様 (T13-2)

全体的な仕様:

- 3次元ベクトルの型 `Vector` 型を定義して利用せよ。 `Vector` 型は以下のように定義してあること。

```
/*構造体テンプレート定義*/
/*3次元ベクトルを表す構造体*/
struct vector
{
    double x; /*x成分*/
    double y; /*y成分*/
    double z; /*z成分*/
};

/*3次元ベクトル型 Vector の定義*/
typedef struct vector Vector;
```

- ベクトルを標準入力より入力する関数 `scan_vector` およびベクトルを標準出力へ出力する関数 `print_vector` を作れ。 `scan_vector` , `print_vector` の各仕様は後の説明を参照。
- 3次元ベクトルの内積を求める関数 `inner_product`, 3次元ベクトルの外積を求める関数 `outer_product` を作れ。関数 `inner_product`, `outer_product` の各仕様は後の説明を参照。
- グローバル変数を用いないこと。
- プログラム全体の入力は, 2つのベクトルを標準入力から行なう。各ベクトル毎に `x成分`, `y成分`, `z成分`の順に標準入力から行なう。
- プログラム全体の出力は, 全て標準出力に行なう。入力された2つの3次元ベクトル (`Vector` 型の値) と, その2つのベクトルの内積 (実数, `double` 値), その2つのベクトルの外積 (3次元ベクトル, `Vector` 型の値) を出力する。スカラー (実数, `double` 値) は小数点以下2桁まで出力する。各3次元ベクトル (`Vector` 型の値) は (`x成分`, `y成分`, `z成分`) の形式で, 各成分を小数点以下2桁まで出力する。実行例参照。
内積はスカラーであり, 外積は3次元ベクトルであることに注意せよ。
- `main` 関数中では `scanf` 文を用いないこと。
- `main` 関数中では, `printf` 文を用いて `Vector` 型の変数の値の表示を行わないこと。

3次元ベクトルの各成分を標準入力から読み込む関数 `scan_vector` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*  
機能説明 :  
3次元ベクトルを標準入力から読み込む関数  
  
仮引数 : なし (void)  
  
戻り値 : 標準入力から読み込んだ 3次元ベクトル  
  
副作用 :  
標準入力から 3つの double 値を読み込み ,  
1 目 double 値を戻り値の x 成分 ,  
2 目 double 値を戻り値の y 成分 ,  
3 目 double 値を戻り値の z 成分とする .  
*/  
Vector scan_vector();
```

- 仮引数はない . すなわち , 仮引数の型は `void` とする .
- 戻り値は , 標準入力より読み込まれた 3次元ベクトル (`Vector` 型の値) とする . 標準入力より 3つの実数 (`double` 値) を読みとり , 1 番目の実数を `x` 成分 , 2 番目の実数を `y` 成分 , 3 番目の実数を `z` 成分とする 3次元ベクトルを返す .
- 関数 `scan_vector` 中では標準出力による出力は行なってはならない . 例えば , `printf` 文による表示を行ってはならない .

3次元ベクトルを標準出力へ出力する関数 `print_vector` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*
機能説明 :
3次元ベクトルを標準出力へ出力する関数

仮引数 :
v : 標準出力へ出力すべき 3次元ベクトル .

戻り値 : なし (void)

副作用 :
仮引数で与えられた 3次元ベクトル v を整形して
標準出力へ出力する .
*/
void print_vector(Vector v);
```

- 仮引数 `v` は標準出力へ出力したい任意の 3次元ベクトル (`Vector` 型の値) とする .
- 戻り値はない . すなわち , 戻り値の型は `void` 型とする .
- 仮引数 `v` で与えられた 3次元ベクトルの各成分を , (`x` 成分 , `y` 成分 , `z` 成分) の形式で標準出力に出力する副作用を持つ . 各成分は小数点以下 2 桁まで出力する .
- 関数 `print_vector` 中では , 3次元ベクトルの値以外の表示を行なってはならない . 特に , “改行” の表示は行なはないこと .
- 関数 `print_vector` 中では標準入力による値の読み込みは行なわない . 例えば , `scanf` 文を用いてはならない .

2つの3次元ベクトルの内積を計算する関数 `inner_product` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*
機能説明:
2つの3次元ベクトルの内積を計算する関数

仮引数:
operand1: 被演算項 1, 内積の左辺の3次元ベクトル .
operand2: 被演算項 2, 内積の右辺の3次元ベクトル .

戻り値:
内積を表す実数値, operand1 * operand2

*/
double inner_product(Vector operand1, Vector operand2);
```

- 仮引数は以下の意味を持つ .
 - 仮引数 `operand1` は被演算項であり, 演算の左辺の値 (3次元ベクトル, `Vector` 型の値) を表す . `operand1` は任意の3次元ベクトルでよい .
 - 仮引数 `operand2` は被演算項であり, 演算の右辺の値 (3次元ベクトル, `Vector` 型の値) を表す . `operand2` は任意の3次元ベクトルでよい .
- 戻り値として, 内積 (`operand1 * operand2`) の値を返す . 内積はベクトルではなくてスカラー (実数, `double` 値) であることに注意せよ .
- 関数 `inner_product` 中では, 標準入出力は行なってはならない . 例えば, `scanf` 文や `printf` 文は用いてはならない .

2つの3次元ベクトルの外積を計算する関数 `outer_product` の仕様:

- 次のプロトタイプ宣言を持つ .

```
/*
機能説明:
2つの3次元ベクトルの外積を計算する関数

仮引数:
operand1:被演算項1, 外積の左辺の3次元ベクトル.
operand2:被演算項2, 外積の右辺の3次元ベクトル.

戻り値:
外積を表す3次元ベクトル, operand1 × operand2
*/
Vector outer_product(Vector operand1, Vector operand2);
```

- 仮引数は以下の意味を持つ .
 - 仮引数 operand1 は被演算項であり, 演算の左辺の値 (3次元ベクトル) を表す . operand1 は任意の3次元ベクトルでよい .
 - 仮引数 operand2 は被演算項であり, 演算の右辺の値 (3次元ベクトル) を表す . operand2 は任意の3次元ベクトルでよい .
- 戻り値として, 外積 (operand1 × operand2) を返す . 外積はの3次元ベクトル (Vector 型の値) であることに注意せよ .
- 関数 outer_product 中では, 標準入出力は行なってはならない . 例えば, scanf 文や printf 文は用いてはならない .