電子 1(木曜クラス)

第 12 回課題 T12

(新しい型と構造体:教科書第 16 章、2006/7/5(木))

基本問題

T12-1:複素数 (本提出期限 2007/7/5(木)17:40、再提出期限 2007/7/26(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

```
b08b0xx@tyy:~/T12/1$./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)
b08b0xx@tyy:~/T12/1$
```

要求仕様 (T12-1)

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，pritnf 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.

```
/*複素数を標準入力から読み込む関数*/
Complex scan_complex();
```

- 仮引数は無い．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.

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.

```
/*複素数を標準出力に出力する関数*/
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，虚部共にも小数点以下 2 桁まで出力する.
- 関数 print_complex 中では改行の表示は行なわない.
- 関数 print_complex 中では標準入力による値の読み込みは行なってはならない．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.

```
/*2 の複素数の和を求める関数*/
Complex add_complex(Complex operand1,Complex prerand2);
```

- 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.

```
/*2 つの複素数の差を求める関数*/
Complex sub_complex(Complex operand1,Complex prerand2);
```

- 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.

```
/*2 つの複素数の積を求める関数*/
Complex mul_complex(Complex operand1,Complex prerand2);
```

- 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.

```
/*2 つの複素数の商を求める関数*/
Complex div_complex(Complex operand1,Complex prerand2);
```

- 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

T12-2:ベクトル
(本提出期限 2007/7/12(木)17:40、再提出期限 2007/7/27(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

```
b08b0xx@tyy:~/T12/2$./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)
b08b0xx@tyy:~/T12/2$
```

要求仕様 (T12-2)

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,doulbe 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,doulbe 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，pritnf 文を用いて Vector 型の変数の値の表示を行なわないこと．

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.

```
/*3 次元ベクトルを標準入力から読み込む関数*/
Vector scan_vector();
```

- 仮引数は無い．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.

```
/*3 次元ベクトルを標準出力へ出力する関数*/
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.

```
/*2 つの 3 次元ベクトルの内積を計算する関数*/
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.

```
/*2 つの 3 次元ベクトルの外積を計算する関数*/
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.