構造体と列挙型

演習問題 15.1　文法事項

1）構造体の定義と構造体変数の宣言について簡単に説明しなさい。

解答例　構造体の定義とは、その構造体が含むメンバなどを定義して、構造体がどのようなデータを保持するかを定めることです。構造体変数の宣言は、構造体の定義に従って、実際にデータを保存する変数領域を確保します。

構造体を図書館のカードに例えると、構造体の定義はカードの書式を決めることに相当し、構造体変数の宣言は実際にカードを準備することに相当します。

また、構造体の定義は、「既に存在するデータ型を組み合わせて、新たなデータ型を作り出すこと」と考えることもできます。構造体変数の宣言は、構造体として新たに作り出されたデータ型の変数を準備することと考えられます。

2）構造体のポインタから、構造体のメンバを操作する 2 つの方法を挙げなさい。

解答例　構造体のポインタ変数 ***ptr** があり、その構造体にメンバ **member** があるとしましょう。このとき、**ptr** が指す構造体変数の **member** にアクセスするためには、ポインタ演算子を用いて、

```
(*ptr).member
```

とする方法と、「**->**」を用いて

```
ptr->member
```

とする方法があります。

3）構造体と関数は、情報を隠蔽する機能があります。それぞれが隠蔽するものは何か簡潔に答えなさい。

解答例　構造体はデータの詳細を隠蔽し、関数は手続き (アルゴリズム) の詳細を隠蔽します。

4）情報の隠蔽の有効性について簡潔に述べなさい。

解答例　構造体や関数を上手く使って情報隠蔽を行うことにより、プログラムの設計者はプログラムをより抽象度が高いレベルで考えることができ、プログラム開発の効率をあげることができます。

もっとざっくばらんに言えば、情報隠蔽によってデータや手続きの詳細が隠蔽され、細かいことを考えなくてよくなるため、プログラムの大きな流れがよく見通せるようになり、大規模なプログラムでも設計しやすくなるということです。

演習問題 15.2

プログラム例15.3に、複素数の引き算と掛け算を行う関数を付け加え、足し算、引き算、掛け算を行えるプログラムを作りなさい。

解答例　解答例を解答プログラム 15.1 に示します。

このプログラムは、プログラム例 15.3 に引き算と掛け算の関数を付け加えただけなので、さほど難しくはないですね。

それよりも、**main()** 関数に注目して下さい。そこには、構造体変数の **aVer** や **bVer** が宣言されていますが、その中身がどんなデータなのかは書かれていません。また、複素数入力の関数や足し算、引き算などの関数の呼び出しが並んでいるだけで、複素数の演算の詳細は書かれていません。ただ、複素数を入力し、演算し、結果を出力していることがけが読み取れます。これが､情報隠蔽の素晴らしい点です。データがどんな構造を持っているのかといった詳細は、構造体の定義の部分に隠蔽されています。各々の演算がどのように行われるのかという手続きの詳細は関数の中に隠蔽されています。このため、**main()** 関数では、おおまかな処理の流れだけを把握していればよいのです。

解答プログラム 15.1　**複素数の演算**

```
/************************************************
    p15_1.c
    複素数の演算　構造体と関数
    Tadaaki Shimizu    2007.01.15
************************************************/
#include <stdio.h>
struct complex {  // 複素数型
    double re;  // 実部
    double im;  // 虚部
};
struct complex  getComplex(void);
struct complex  addComplex(struct complex, struct complex);
struct complex  subComplex(struct complex, struct complex);
struct complex  multComplex(struct complex, struct complex);
void            printComplex(struct complex);

main()
{
    struct complex aVer, bVer;  // 入力される複素数
    struct complex answer;      // 答えの複素数

    aVer = getComplex();
    bVer = getComplex();

    answer = addComplex(aVer,bVer);
    printf("足し算： ");
    printComplex(answer);
    answer = subComplex(aVer,bVer);
    printf("引き算： ");
    printComplex(answer);
    answer = multComplex(aVer,bVer);
```

```
    printf("掛け算： ");
    printComplex(answer);

    exit(0);
}

/************************************************
    関数名       : getComplex() 複素数の入力
    引数         : なし
    返り値       : struct complex 入力された複素数
************************************************/
struct complex getComplex(void)
{
    struct complex inputVer;  // 複素数入力

    printf("複素数：実部入力 >> ");
    scanf("%lf", &inputVer.re);
    printf("複素数：虚部入力 >> ");
    scanf("%lf", &inputVer.im);

    return(inputVer);
}

/************************************************
    addComplex() 複素数の足し算
    引数         : struct complex aArg,bArg 足し算の演算項
    返り値       : struct complex 足し算の答え
************************************************/
struct complex addComplex(struct complex aArg,
                          struct complex bArg)
{
    struct complex answer;  // 足し算の答え

    answer.re = aArg.re + bArg.re;  // 実部の足し算
    answer.im = aArg.im + bArg.im;  // 虚部の足し算

    return(answer);
}

/************************************************
    subComplex() 複素数の引き算
    引数         : struct complex aArg,bArg 引き算の演算項
    返り値       : struct complex 引き算の答え
************************************************/
struct complex subComplex(struct complex aArg,
                          struct complex bArg)
{
    struct complex answer;  // 引き算の答え

    answer.re = aArg.re - bArg.re;  // 実部の引き算
    answer.im = aArg.im - bArg.im;  // 虚部の引き算

    return(answer);
}
```

```
/************************************************
    multComplex() 複素数の掛け算
    引数         : struct complex aArg,bArg 掛け算の演算項
    返り値       : struct complex 掛け算の答え
************************************************/
struct complex multComplex(struct complex aArg,
                           struct complex bArg)
{
    struct complex answer;  // 掛け算の答え

    answer.re = (aArg.re * bArg.re) - (aArg.im * bArg.im);
    answer.im = (aArg.re * bArg.im) + (aArg.im * bArg.re);

    return(answer);
}

/************************************************
    printComplex() 複素数の出力
    引数         : struct complex outData  出力する複素数
    返り値       : なし
************************************************/
void printComplex(struct complex outData)
{
    printf("%lf + %lf i\n", outData.re, outData.im);
    return;
}
```

演習問題 15.3

身体測定のデータ処理を行いたい。氏名と身長，体重のデータを次の構造体の配列で扱い、身長と体重の平均と最大，最小を求めるプログラムを作りなさい。ただし、身体測定に参加する人数は20人以下とします。

```
struct personalData {   // 身体測定データ構造体
    char name[20];   // 氏名
    double hgt;      // 身長
    double wgt;      // 体重
};

struct personalData psnData[20];   // 身体測定データ配列
```

解答例　解答例を解答プログラム 15.2 に示します。

この演習問題は、凝れば色々と考える部分もありますが、最低限のプログラムとしました。注目する点は、まず、演習問題 15.2 と同様に **main()** 関数が非常に単純であることです。これは、もちろん **searchMaxHight()** 関数などの関数に肝心の手続きが隠蔽されているからです。

初心者にとって、このプログラムの嫌な所は、構造体の配列が用いられて点でしょう。しかし、よく見てみれば、構造体として定義した新しいデータ型も、**int** や **double** といった一般的なデータ型と全く同じように配列を作ることができることがよくわかるはずです。

プログラムを作るときに、情報隠蔽を上手くやるコツとして、次のことを挙げておきます。

1) 構造体の操作には、専用の関数を作る
2) 構造体のメンバには、専用関数の中だけでアクセスする

この方針によって設計すると、**main()** 関数などの上位の関数では、構造体のメンバに直接アクセスする必要が無くなります。言い換えると、構造体の中にどんなメンバが存在するのか詳細は知らなくても (忘れてしまっても)**main()** 関数を作ることができます。解答プログラム 15.2 の **main()** 関数にも、構造体のメンバは一切現れていません。これが、**main()** 関数が非常に単純なものになっている最大の理由なのです。

解答プログラム 15.2　身体測定データ処理

```
/************************************************
    p15_2.c
    身体測定データ処理
    Tadaaki Shimizu    2007.01.16
************************************************/
#include <stdio.h>
struct personalData {   // 身体測定データ構造体
    char name[20];  // 氏名
    double hgt;     // 身長
    double wgt;     // 体重
};
struct personalData getPersonalData(void);
double searchMaxHight(struct personalData [], int);
double searchMinHight(struct personalData [], int);
double searchMaxWeight(struct personalData [], int);
double searchMinWeight(struct personalData [], int);

main()
{
    struct personalData psnData[20];   // 身体測定データ配列
    int    numOfPsn; // 人数
    double data;     // 最大値等のデータ
    int    i;

    printf("人数を入力 : ");
    scanf("%d", &numOfPsn);
    for(i = 0; i < numOfPsn; i++) {
        printf("%d 人目入力\n", i+1);
        psnData[i] = getPersonalData();
    }
```

```
    data = searchMaxHight(psnData, numOfPsn);
    printf("最大身長 : %lf [cm]\n", data);
    data = searchMinHight(psnData, numOfPsn);
    printf("最小身長 : %lf [cm]\n", data);
    data = searchMaxWeight(psnData, numOfPsn);
    printf("最大体重 : %lf [kg]\n", data);
    data = searchMinWeight(psnData, numOfPsn);
    printf("最小体重 : %lf [kg]\n", data);
    exit(0);
}

/************************************************
    関数名       : getPersonalData() データの入力
    引数         : なし
    返り値       : struct personalData 身体測定データ
************************************************/
struct personalData getPersonalData(void)
{
    struct personalData inputData;

    printf(" 名前 : ");
    scanf("%s", inputData.name);
    printf(" 身長 : ");
    scanf("%lf", &inputData.hgt);
    printf(" 体重 : ");
    scanf("%lf", &inputData.wgt);
    return inputData;
}

/************************************************
    関数名       : searchMaxHight() 身長の最大値を探す
    引数         : struct personalData psnData[] 身体測定データ
                   int numOfPsn 人数
    返り値       : double 身長の最大値
************************************************/
double searchMaxHight(struct personalData psnData[], int numOfPsn)
{
    double maxHight; // 身長の最大値
    int    i;

    // 最大値探索
    maxHight = 0.0; // とりあえずの最大値
    for(i = 0; i < numOfPsn; i++)
        if(maxHight < psnData[i].hgt)
            maxHight = psnData[i].hgt;
    return(maxHight);
}

/************************************************
    関数名       : searchMinHight() 身長の最小値を探す
    引数         : struct personalData psnData[] 身体測定データ
                   int numOfPsn 人数
    返り値       : double 身長の最小値
************************************************/
```

```
double searchMinHight(struct personalData psnData[], int numOfPsn)
{
    double minHight; // 身長の最小値
    int    i;

    // 最小値探索
    minHight = 1e10; // とりあえずの最小値
    for(i = 0; i < numOfPsn; i++)
        if(minHight > psnData[i].hgt)
            minHight = psnData[i].hgt;
    return(minHight);
}

/************************************************
    関数名       : searchMaxWeight() 体重の最大値を探す
    引数         : struct personalData psnData[] 身体測定データ
                   int numOfPsn 人数
    返り値       : double 体重の最大値
************************************************/
double searchMaxWeight(struct personalData psnData[], int numOfPsn)
{
    double maxWeight; // 体重の最大値
    int    i;

    // 最大値探索
    maxWeight = 0.0; // とりあえずの最大値
    for(i = 0; i < numOfPsn; i++)
        if(maxWeight < psnData[i].wgt)
            maxWeight = psnData[i].wgt;
    return(maxWeight);
}

/************************************************
    関数名       : searchMinWeight() 体重の最小値を探す
    引数         : struct personalData psnData[] 身体測定データ
                   int numOfPsn 人数
    返り値       : double 体重の最小値
************************************************/
double searchMinWeight(struct personalData psnData[], int numOfPsn)
{
    double minWeight; // 体重の最小値
    int    i;

    // 最小値探索
    minWeight = 1e10; // とりあえずの最小値
    for(i = 0; i < numOfPsn; i++)
        if(minWeight > psnData[i].wgt)
            minWeight = psnData[i].wgt;
    return(minWeight);
}
```