解答編　第 3 章

条件による分岐 1

演習問題 3.1　文法事項

1）C 言語で用いる関係演算子を 6 つあげなさい。

解答例　`==, !=, <, <=, >, >=`

2）C 言語で用いる論理演算子を 3 つあげなさい。

解答例　`&&, ||, !`

3）関係演算子や論理演算子を用いて作られた論理式は、どのような値をとりますか。

解答例　論理式が偽なら整数値の `0`、真ならば `0` 以外の値 (普通は `1`)。

4）**`if`** 文や **`else`** 文に従属する文が複数ある場合、従属する文の範囲を示すためにどのようにしますか。

解答例　{}(中カッコ) で、従属する複数の文を囲む。ちなみに，中カッコで複数の文をまとめたものを複文と呼び，あたかも 1 つの文のように扱われます.

5）`if` 文や `else` 文に従属する文を明確に示すために、書式上どのような工夫をすべきですか。

解答例　字下げ (インデント) を使って、従属する範囲が直観的にわかるような書式とするべきです。

演習問題 3.2

プログラム例3.9で、文法的に誤っている部分を探しなさい。また、文法的な誤りではないが、読みやすさの点で問題があると思われる部分を探しなさい。また、それぞれ、何故誤っているか、何故問題があるかを答えなさい。

解答例　1) 文法的な誤り

- 2 番目の **`scanf()`** 関数の呼び出しに誤りがあります。変数 **`secondNum`** の前に **`&`**(アンパサンド) が欠けているため正常に動作しません。
  (この誤りは、厳密には文法誤りでないため、コンパイラはエラーメッセージを発しません。)
- **`if`** 文に誤りがあります。従属範囲を示す中カッコが閉じていません。
- 最後から 4 行目の **`printf()`** 関数に誤りがあります。書式文字列において、**`"`**(ダブルクォート) が閉じていません。

2) 読みやすさの点で問題がある箇所

- 最初の変数宣言に問題があります。これにもコメント文を付けることが望ましい。
- **`else`** 文に問題があります。従属する 2 つの文の字下げ位置が悪いです。読みやすさの点から、**`if`** 文と合わせた字下げ位置を採用するのが望ましい。

演習問題 3.3

支払い金額を入力すると、最小の貨幣数で支払う方法を教えてくれるプログラムを作りなさい。ただし、支払い金額は、10000円以下とし、使える貨幣は、1000円札と、500円、100円、50円、10円、5円、1円の各硬貨とします。

解答例　考え方は簡単です。最小の貨幣数で支払うには、なるべく高額な貨幣を沢山使えばよいはずです。例えば、2800 円を支払う場合、まず 1000 円札で支払います。1000 円札が何枚使えるかを知るには、2800 を 1000 で割ります。割り算の結果は、2 余り 800 となり、1000 円札を 2 枚支払って、残りの支払い額が 800 円となることがわかります。このような操作を価値の高い貨幣から順に繰り返せば、最小の貨幣数を求めることができます。

以上の考え方で作ったプログラムを、解答プログラム 3.1 に示します。プログラムは、単純なので詳しい解説は省きますが、支払い額の入力の際に値のチェックを行って、エラー処理をすることを忘れないようにしましょう。

解答プログラム 3.1 は、各貨幣について同じような処理が並んでいて格好良くありません。繰り返し処理の構文を学ぶと、もう少しスマートなプログラムを書くことができますので、後でもう一度この課題にチャレンジするとよいでしょう。

解答プログラム 3.1　支払い貨幣を調べるプログラム

```
/************************************
    p3_1.c
    支払い貨幣を調べるプログラム
    Tadaaki Shimizu  2006.11.08
************************************/
#include <stdio.h>

main()
{
    int payment;      // 支払い額
    int numOfCoins;   // 硬貨の枚数
    int remainder;    // 支払い額の残り
    int total;        // 貨幣の総数


    // 支払額の入力
    printf("支払い額の入力 (0-10000) >>");
    scanf("%d", &payment);
    if(payment < 0 || payment > 10000) {   // 入力エラー処理
        printf("ERROR: 入力額が範囲外です。\n");
        exit(0);
    }
    // 支払い貨幣の計算
    total = 0; // 貨幣の総数の初期化
    numOfCoins = payment / 1000;
    if(numOfCoins != 0) {
        printf("1000円 : %d枚\n", numOfCoins);
        remainder = payment % 1000;
        total = total + numOfCoins;
    }
```

```
    numOfCoins = remainder / 500;
    if(numOfCoins != 0) {
        printf("500円 : %d枚\n", numOfCoins);
        remainder = remainder % 500;
        total = total + numOfCoins;
    }
    numOfCoins = remainder / 100;
    if(numOfCoins != 0) {
        printf("100円 : %d枚\n", numOfCoins);
        remainder = remainder - numOfCoins * 100;
        total = total + numOfCoins;
    }
    numOfCoins = remainder / 50;
    if(numOfCoins != 0) {
        printf("50円 : %d枚\n", numOfCoins);
        remainder = remainder % 50;
        total = total + numOfCoins;
    }
    numOfCoins = remainder / 10;
    if(numOfCoins != 0) {
        printf("10円 : %d枚\n", numOfCoins);
        remainder = remainder % 10;
        total = total + numOfCoins;
    }
    numOfCoins = remainder / 5;
    if(numOfCoins != 0) {
        printf("5円 : %d枚\n", numOfCoins);
        remainder = remainder % 5;
        total = total + numOfCoins;
    }
    if(remainder != 0) {
        printf("1円 : %d枚\n", remainder);
        total = total + remainder;
    }
    printf("%d円を支払うための貨幣の総数は %d枚でした。\n", payment, total);
    exit(0);
}
```

[実行例]

```
% ./p3_1
支払い額の入力 (0-10000) >>1999
1000円 : 1枚
500円 : 1枚
100円 : 4枚
50円 : 1枚
10円 : 4枚
5円 : 1枚
1円 : 4枚
1999円を支払うための貨幣の総数は 16枚でした。
%
```

演習問題 3.4

10進数 i を入力し、$f(i) = |i - 10| + |i + 5|$ を計算するプログラムを作りなさい。ただし、プログラム中で iなどという変数名を用いないようにしなさい。

解答例　解答プログラム 3.2 のように作ってみました。

見ての通り、変数 **inputData** に i の値を入力し、まず **inputData - 10** を計算して変数 **answer** に格納します。この結果がもし負であれば、正になるように符号を反転します。これが絶対値の演算に相当します。次に、**inputData + 5** を計算して変数 **tmp** に格納します。先と同じ方法で **tmp** の絶対値を計算します。最後に、**answer** に **tmp** の値を足すことで $f(i)$ の値が求まります。

ここで、変数 **tmp** は、計算のためにほんの一時的に使われる変数です。このような変数をテンポラリ変数と呼びます。テンポラリ変数は、計算や処理の中間的な値を少しの間記録しておくためのメモ書きの役割をする変数です。テンポラリ変数を上手く使うと、プログラムを楽に書くことができます。テンポラリ変数には、いつでも同じ名前を付けるように決めておくと、自分でプログラムを読むときにすぐにそれとわかって便利です。著者は、**tmp** や **temp** という名前を良く使います。テンポラリ変数は、ちょっとしたメモのための変数なので、プログラムの広範囲にわたって (何行にもわたって) 値を保持するような使い方は避ける方が無難です。

解答プログラム 3.2　演習課題 3.4 のプログラム

```
/************************************
    p3_2.c
    |i - 10| + |i + 5| を計算
    Tadaaki Shimizu  2006.11.08
************************************/
#include <stdio.h>

main()
{
    int inputData;    // 入力された数値
    int answer;       // 計算結果
    int tmp;          // 計算用テンポラリ変数

    // i の入力
    printf("i の入力 ( 整数 ) >>");
    scanf("%d", &inputData);

    answer = inputData - 10;
    if(answer < 0) answer = - answer;
    tmp    = inputData + 5;
    if(tmp < 0) tmp = - tmp;
    answer = answer + tmp;

    printf("|i - 10| + |i + 5| = %d\n", answer);
    exit(0);
}
```

演習課題 3.4 の場合、$f(i)$ を少し書き換えて、次のように記述することが可能です。

$$f(i) = 5 - 2i \qquad (i < -5)$$
$$f(i) = 15 \qquad (-5 \leq i < 10)$$
$$f(i) = 2i - 5 \qquad (i \geq 10)$$

このように式を変形すると、プログラムを、解答プログラム 3.3 のように作ることができます。解答プログラム 3.3 はテンポラリ変数が不要で、解答プログラム 3.2 よりもスマートに見えます。しかし、必ずしも解答プログラム 3.3 の方がよいとは限りません。何故なら、解答プログラム 3.3 を作るためには、解答プログラム 3.2 を作るよりも、より多く考える必要があり、時間がかかるからです。色々考えて解答プログラム 3.3 を作っている間に、解答プログラム 3.2 を作り終えて計算結果を得ることができます。プログラムを作るときには、プログラム自体のことだけでなく、問題を解くプロセス全体を見渡したバランス感覚が大切です。

解答プログラム 3.3　演習課題 3.4 のプログラム (別解)

```
/************************************
    p3_3.c
    |i - 10| + |i + 5|を計算
    Tadaaki Shimizu  2006.11.09
************************************/
#include <stdio.h>

main()
{
    int inputData;    // 入力された数値
    int answer;       // 計算結果

    // i の入力
    printf("i の入力 ( 整数 ) >>");
    scanf("%d", &inputData);

    if(inputData < -5)
        answer = 5 - 2 * inputData;
    else if(inputData >= -5 && inputData < 10)
        answer = 15;
    else
        answer = 2 * inputData - 5;

    printf("|i - 10| + |i + 5| = %d\n", answer);
    exit(0);
}
```

演習問題 3.5

1）プログラム例 3.5 は、入力する数値によっては実行時に問題を起こします。どのような問題か考えなさい。また、わざと問題を起こす入力を試してみなさい。

(ヒント：割り算が実行できない場合がありませんか？)

解答例　数を 0 で割る事はできません。C 言語では､0 による割り算は定義されていません。定義されていないというのはなかなか厄介で、コンパイラや実行環境によって動作が異なる可能性があるということです。

著者の環境で試してみると、下記のように、**Turbolinux** 上で **gcc** バージョン **3.3.1** を使った場合は、プログラムが異常停止 **(core dumped)** しました。

［実行例］

```
% ./ex3_5
Input Integer Number 1 > 2
Input Integer Number 2 > 0
Floating point exception (core dumped)
%
```

一方、**Mac OSX 10.4.8** 上で、**gcc version 4.0.1** を使った場合、0 で割った割り算は 0 になるようです (実数の割り算は 0 にはなりません)。

```
% ./ex3_5
Input Integer Number 1 > 2
Input Integer Number 2 > 0
div = 0
mod = 2
%
```

とは言え、0 による割り算はプログラムの異常終了につながることがほとんどのようです。

2）この問題を回避できるように、プログラム例 3.5 を書き直しなさい。

解答例　この問題を本質的に回避する方法はありません｡出来ないものは出来ないからです。そこで、0 による割り算が起きる場合にはエラーメッセージを発して、処理自体を行わないようにします。解答プログラム 3.4 にその方法を示します。

解答プログラム 3.4 は、プログラム例 3.5 をなるべく変えないようにエラー処理を付け加えました。処理全体を見直すと、プログラムをもう少しスマートにすることも出来ますので、興味のある方はチャレンジしてみて下さい。

解答プログラム 3.4　**プログラム例 3.5 の改良　～ 0 による割り算の回避**

```
/************************************************
    p3_4.c    ex3_5 の改良版
    入力した 2 整数の商と余りを求めるプログラム
        大きい方の数を小さい方の数で割る
    Tadaaki Shimizu     2006.11.09
************************************************/
#include <stdio.h>

main()
{
    int firstNum, secondNum;    //  入力される 2 整数
    int div, mod;               //  商と余り

    // データの入力
    printf("Input Integer Number 1 > ");
    scanf("%d", &firstNum);
    printf("Input Integer Number 2 > ");
    scanf("%d", &secondNum);

    // 商と余りを求める処理
    if(firstNum > secondNum) {
        if(secondNum == 0) {
            printf("ERROR: 0 による割り算は出来ません。\n");
            exit(0);
        }
        div = firstNum / secondNum;
        mod = firstNum % secondNum;
    }
    else {
        if(firstNum == 0) {
            printf("ERROR: 0 による割り算は出来ません。\n");
            exit(0);
        }
        div = secondNum / firstNum;
        mod = secondNum % firstNum;
    }
    printf("div = %d\n", div);
    printf("mod = %d\n", mod);
    exit(0);
}
```