プログラミング及び実習 I ・第 7 回

2024 年 7 月 18 日

今回の内容

7.1 繰り返しの処理 (続き)

今回は、引き続き、繰り返しの処理に関係するプログラミングの考え方について勉強します。

フラグ 次のようなプログラムを作ることを考えます。

> キーボードから正の整数を入力すると、それが素数であるかどうかを判定して表示するプログラム (`prime.c`)

素数とは、1 と自分自身以外の正の整数では割り切れないような 2 以上の整数のことです。このプログラムをコンパイルして実行したら、次の例のように動かなければなりません。

素数でない数が入力された場合

```
正の整数 => 38
38は素数ではありません
```

素数が入力された場合

```
正の整数 => 37
37は素数です
```

正の整数が素数であるかどうかを判定するためには、次のような仕事を行えばよいはずです。

1. 正の整数をキーボードから入力してもらい、それを変数 (たとえば `n`) に格納しておく。
2. 2 から `n`−1 までの整数で割ったときの余りをそれぞれ計算する。
3. `n` が 1 以下であるか、あるいは (手順 2 の) いずれかの数で割ったときの余りが 0 ならば「素数でない」と出力し、どの数で割ったときの余りも 0 でなければ「素数です」と出力する。

この内、手順の 2 は繰り返しの処理 (`while` 文) になりますし、手順の 3 は場合分けの処理 (`if` 文) になります。では、「(手順 2 の) いずれかの数で割ったときの余りが 0」という条件をどうやって調べたらよいのでしょうか? ある変数の値がある定数に等しいとか、ある定数より大きいとかいった類の条件であれば、それをそのまま `if` 文の条件式の部分に書けばよいわけですが、「いずれかの数で割ったときの余りが 0」という条件をそのまま `if` 文の条件式の部分に書くことはできません。「割ったときの余りが 0」ということは、あくまで 2 から `n`−1 までのそれぞれの整数に対して (手順 2 の繰り返しの処理の中で) 分かるだけです。このような場合、次のように考えてプログラムを作るとうまく行きます。

- 繰り返しの処理を始める前に、ある変数 (たとえば `found`) に 0 を格納しておく。
- 手順 2 の繰り返しの処理では、それぞれの数で割ったときの余りが 0 だった場合に、その印として変数 `found` に 1 を代入する。

- 手順3の段階で、変数 found に格納されている値が1の場合は「いずれかの数で割ったときの余りが0」と解釈し、0の場合は「どの数で割ったときの余りも0でない」と解釈する。

どの数で割っても割りきれない (余りが0でない) 場合は、変数 found には繰り返しの処理を始める前に代入した0が格納されたままのはずですし、いずれかの数で割りきれた (余りが0だった) のなら、そこで変数 found の値は1に変ってしまっているはずです。

メモ

変数 found は「それまでに割りきれた数があるかどうか」という情報を保持していると考えることができます。変数 found が1ならそのような数があったわけですし、0ならまだそのような数は見つかっていないということです。この found のように、何かの仕事を繰り返して (続けて) ゆく中で、ある特定の状況がすでに起ったかどうか (ある特定の条件がすでに成り立ったかどうか) を示すために使われる変数を「**フラグ**(旗)」と呼びます。その特定の状況が起ったときに変数の内容を書き換えることを「フラグを立てる」といいます。この例では、まず最初に found に0を代入して (フラグを降ろして) おき、「ある数で n が割りきれた」という状況でフラグを立てています。このフラグは、while 文による繰り返しの処理がすべて完了した時に調べられ、そこで場合分けの処理を行うわけです。

次は、以上のような考えに沿って書かれたプログラムです。

prime.c

```
#include <turtle.h>

main ()
{
    int n, k, found;

    tPrintf("正の整数 => ");
    tScanf("%d", &n);
    found = 0;
    k = 2;
    while (k < n) {
        if (n % k == 0)
            found = 1;
        k = k+1;
    }
    if (n <= 1 || found == 1)
        tPrintf ("%dは素数ではありません\n", n);
    else
        tPrintf ("%dは素数です\n", n);
}
```

素数でない数が入力された場合

```
正の整数 => 38
38は素数ではありません
```

素数が入力された場合

```
正の整数 => 37
37は素数です
```

繰り返しの処理の途中終了　prime.c のプログラムの行っている仕事をよく観察すると、一部無駄な仕事を行っていることが分かります。2 から順に $n-1$ までの数で n を割ってみていますが、一度ある数で割りきれた (フラグを立てた) のなら、それ以上の数についてさらに調べていく必要は無いはずです。このような考えに基づいて繰り返しの条件を書き換え、以下のようなプログラムにすることもできます。

```
found = 0;
k = 2;
while (found == 0 && k < n) {
    if (n % k == 0)
        found = 1;
    k = k+1;
}
```

つまり、フラグが立っていない (まだ割り切れる数が見つかっていない) 場合に限って繰り返しを続けるわけです。

メモ

break 文による繰り返しからの脱出　先の例では while 文の条件式を工夫することで、不要な繰り返しの処理を取り除きましたが、C ではもっと直接的な方法で繰り返しの処理の途中終了を行うことができます。次のプログラムを見てください。

```
found = 0;
k = 2;
while (k < n) {
    if (n % k == 0) {
        found = 1;
        break;
    }
    k = k+1;
}
```

このプログラムでは、while 文の条件式は prime.c と変っていません。変っているのは n が k で割りきれたときに実行される if 文の中です。変数 found のフラグを立てているのは同じですが、それに引き続いて break; という文を実行しています。

この break は if や while と同じように C の予約語の 1 つで、繰り返しの処理などを途中で終了するときに用います。while 文の中で break; が実行されると、そこで while 文全体の実行が終了します。そこで繰り返しの処理が終るわけです。二重三重の while 文の中で break; が実行された場合は、一番内側の (つまりその break 文の 1 つ外側の) while 文の実行が終了します。

prime.c を break 文を使ってこのように書き換えると、フラグとして使っていた found という変数も必要が無くなっていることに気づきます。変数 k を 2 から順に $n-1$ まで動かして、その数で n が割りきれるかどうかを調べているわけですから、最後まで割りきれなかった場合には k の値が n と等しくなって繰り返しの処理が終っているはずですし、もし途中の数で割りきれてしまったのなら、k は n に達する前に (つまり k < n が成り立っている状態で) 繰り返しの処理が (break 文によって) 終了しているはずです。ですから while 文の後で k < n が成り立っているかどうかを調べることで「いずれかの数で割ったときの余りが 0」かどうかを判定することができます。このような考えに基づいて prime.c を書き直すと次のようなプログラムになります。

smartprime.c

```
#include <turtle.h>

main ()
{
    int n, k;

    tPrintf("正の整数 => ");
    tScanf("%d", &n);
    k = 2;
    while (k < n) {
        if (n % k == 0)
            break;
        k = k+1;
    }
    if (n <= 1 || k < n)
        tPrintf ("%dは素数ではありません\n", n);
    else
        tPrintf ("%dは素数です\n", n);
}
```

素数でない数が入力された場合

正の整数 => 38
38は素数ではありません

素数が入力された場合

正の整数 => 37
37は素数です

この smartprime.c も prime.c と同じように正しく素数かどうかを判定して出力してくれます。

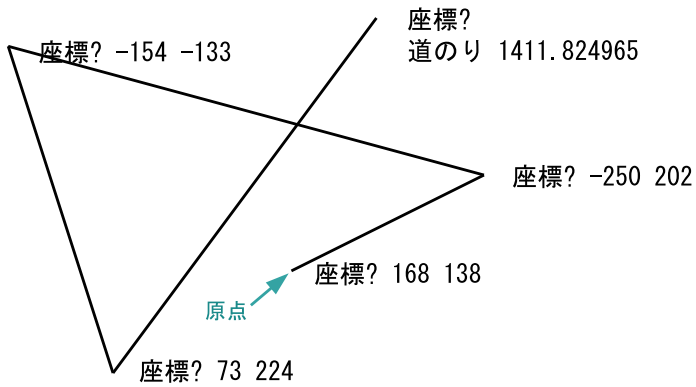
メモ

break 文の応用　break 文はいろいろと応用が効きます。つぎの route.c は、tScanf を使ってキーボードから x 座標と y 座標の値を読み込み、カメを指定された点に次々と移動させていくプログラムです。C 言語では、繰り返しの条件として ( ) の中に定数の 1 を書いておくと、while 文は永遠に繰り返しを続けるようになります。tScanf という関数は、戻り値としてキーボードから読み取ることのできたデータの個数を整数値として返しますので、この数が 2 でない場合に、break 文を使って繰り返しを終了し、これまでの移動の道のり (合計) を表示するようにしています。

route.c

```
 1 #include <turtle.h>
 2
 3 main ()
 4 {
 5     double x, y, sum;
 6
 7     sum = 0;
 8     while (1) {
 9         tPrintf("座標? ");
10         if (tScanf("%lf%lf", &x, &y) != 2)
11             break;
12         sum = sum + tMoveTo(x, y);
13     }
14     tPrintf("道のり %f\n", sum);
15 }
```

下図は、このプログラムを起動後、キーボードから「168 138」、「-250 202」、「-154 -133」、「73 244」の順に入力し、最後に、何も入力せずに Enter キーを押した場合の実行例です。



tScanf の関数呼び出しによって表示された箱に何も入力せずに Enter キーを押すと、戻り値は 0 となりますので、プログラム 11 行目の break 文が実行されて、while 文による繰り返しの処理がそこで終了します。

メモ

7.2 演習問題

1. つぎのようなプログラム comehere.c を作成し、コンパイル、実行して、正しく動作することを確認しなさい。

(1) まず、(100, 80)、(100, 120)、(200, 120)、(200, 80) の 4 つの点を頂点とする長方形を描き、その中に「Come here」という文字列を書く。

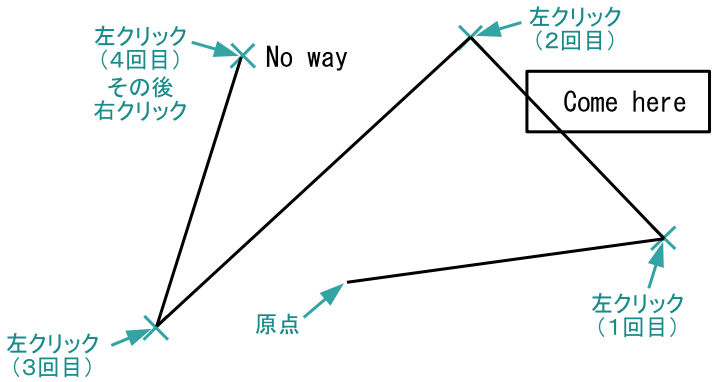
(2) ペンを上げて、原点に戻る。

(3) 左クリックされる度に、クリックされた点に向けて線分を描く、という仕事を繰り返す。

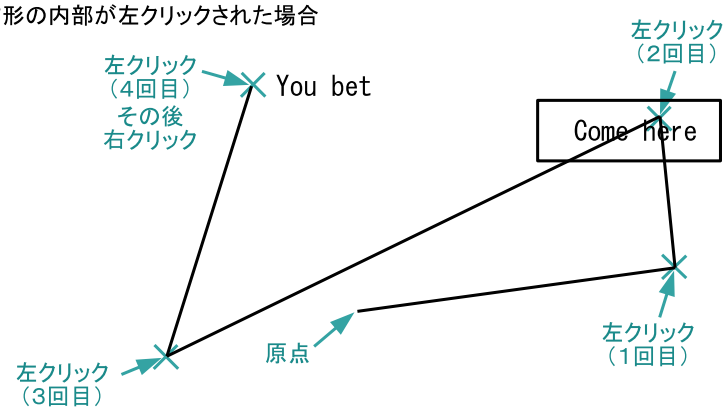
(4) 左ボタン以外でクリックされたら (3) の繰り返しをやめて、それまでに (1) の長方形の内部 (辺上は含まない) が左クリックされたことがあったのなら「You bet」という文字列を、なかったのなら「No way」という文字列を (その場で) 表示する。

ただし、(1) の「Come here」という文字列は、カメを (100, 100) に位置させて書いてください。

長方形の内部が左クリックされなかった場合

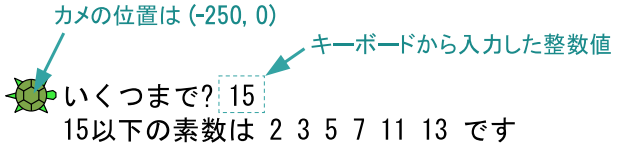


長方形の内部が左クリックされた場合

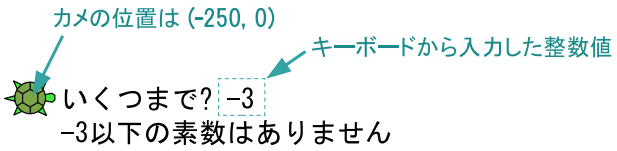


ヒント: (1) の長方形の内部がクリックされたかどうかを、フラグを使って記憶するようにしましょう。

2. キーボードから整数 n を入力すると、n 以下の素数を次の実行例のように出力するプログラム primes.c を作り、コンパイル、実行して、正しく動作することを確認しなさい。ただし、たくさんの素数を表示できるように、まず、カメを $(-250,\ 0)$ の位置に (ペンを上げた状態で) 移動してから始めるようにしてください。このプログラムが表示する「いくつまで?」の右や、「2」、「3」、「5」、... などの素数の両側には半角のスペースが 1 つあります。



また、キーボードから入力されたのが 1 以下の整数であった場合、つぎのように「... 以下の整数はありません」と表示するようにしなさい。



ヒント: 2 から n までの各自然数 m に対して、順に m が素数かどうかを判定し、素数であれば m の値を表示していきましょう。例題の `smartprime.c` のやり方で、ある数が素数であるかどうかを判定することができます。全体は 2 重の繰り返しの処理になるはずです。m を 1 ずつ増やしていく繰り返しの中に、2 から $m-1$ までの各自然数 k で m が割りきれるかどうかを調べていく繰り返しがあることになります。

7.3 今回の実習内容

1. プリントをもう一度読み返しましょう。`prime.c`、`smartprime.c`、`route.c` の 3 つの例題プログラムについては、それぞれテキストエディタを使ってソースファイルを作成し、コンパイル、実行して、正しく動作することを確認してください。プログラムが完成したら「課題の提出と確認」の Web ページから提出してください。
2. 演習問題に取り組みましょう。それぞれのプログラムが完成したら、「課題の提出と確認」の Web ページからの提出を忘れないでください。
3. クイズに答えてください。前回までと同様に「課題の提出と確認」の Web ページで「第 7 回クイズ」を選択し、「送信」のボタンをクリックしてクイズに答えてください。

プログラミング及び実習 I ・第 7 回・終わり