

第 4 章 繰り返し処理

同じような処理を繰り返し実行したい場合は、「for」文もしくは「while」文を用います。それぞれについて、使い方を見ていきましょう。

4.1 for 文

for 文は、ある処理を指定した回数だけ繰り返す際に、よく用いられます。英語で「3 年間」などを表すときに「for three years」と言いますが、for 文の for はこれと同じ意味を持ちます。for 文の書式は以下に示すとおりです。

```
for ( 初期化 ; 条件式 ; 次の処理 ) {  
    // 繰り返し実行する処理  
}
```

このように、for 文では繰り返す回数を「初期化」「条件式」「次の処理」の 3 つの要素で表します（それぞれの要素はセミコロン “;” で区切ります）。文章ではプログラムの流れのイメージが湧きにくいと思いますので、実際のコード例を見てみましょう。以下のソースコードは、0 から 9 までの数字を printf 関数で表示するプログラムです。

例 4-1 数字の 0 から 9 を for 文で表示

```
1 #include <stdio.h>  
2  
3 int main(void) {  
4     int i;  
5     for (i = 0; i < 10; i++) {  
6         printf("%d\n", i);  
7     }  
8     return 0;  
9 }
```

4 行目： 繰り返し回数を数えるための整数型変数 *i* を宣言しています（*i* は integer もしくは index の頭文字を意味します）。

5 行目： for 文による繰り返し処理です。

「初期化」 変数 *i* に 0 を入れて初期化しています。

「条件式」 変数 *i* が 10 未満の間、波括弧の処理を繰り返します。

「次の処理」 波括弧の処理が終わったら、繰り返す前に実行する処理です。ここでは「*i*++」（インクリメント演算）を用いて、*i* に 1 を加算しています。

第4章 繰り返し処理

6 行目： 繰り返す処理の内容です。printf 関数で i の値を表示しています。

例 4-1 実行結果

```
$ ./a
0
1
2
3
4
5
6
7
8
9
```

ここでは 0 から 9 までの数字を表示しましたが、1 から 10 までの数字を表示したい場合は、以下のように表示する際に変数 i に 1 を加えてあげます。

数字の 1 から 10 を for 文で表示する例①

```
int i;
for (i = 0; i < 10; i++) {
    printf("%d\n", i + 1);
}
```

また、変数の値自体に 1 から 10 を入れたい場合、もしくは繰り返しの範囲を 1 から 10 と明示的に表したい場合は、for 文の「初期化」と「条件式」を以下のように修正します。

数字の 1 から 10 を for 文で表示する例②

```
int i;
for (i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
```

もう一つ、簡単で実用的な for 文の例を見てみましょう。次のソースコードは、1 から 10 までの整数の和を求めるプログラムです。

例 4-2 正整数の 1 から 10 までの和を計算

```

1  #include <stdio.h>
2
3  int main(void) {
4      int i, sum = 0;
5      for (i = 1; i <= 10; i++) {
6          sum += i;
7      }
8      printf("1 から 10 までの和は %d です。\\n", sum);
9      return 0;
10 }
```

4 行目で、計算結果を格納するための変数 sum を用意し、6 行目で i の値を sum に加算していきます。for 文で i は 1 から 10 になるまで繰り返されるので、sum には 1 から 10 までの和が入ります。

例 4-2 実行結果

```
$ ./a
```

```
1 から 10 までの和は 55 です。
```

演習 4-1 (ex4-1.c)

正の整数値の入力を受け取り、その約数をすべて表示するプログラムを作ってください。余力があれば、約数の個数と合計値も表示してみましょう。

ヒント：剰余（割った余り）が 0 であるかどうかで、約数かどうかを判定できます。

演習 4-1 実行例

```
$ ./a
```

```
正整数を入力して下さい: 27
```

```
27 の約数は、
```

```
1
```

```
3
```

```
9
```

```
27
```

```
の 4 個で、合計値は 40 です。
```

for 文の入れ子

for 文は入れ子の形でもよく使われます。入れ子はネスト (Nesting) とも言われ、プログラムの構造が再帰的に繰り返されることを表します。例えば、for 文の処理の中にまた for 文があれば、それは入れ子構造です。以下の for 文 n 入れ子構造を持つプログラムを見て、どのような結果になるか予想してみましょう。

例 4-3 入れ子による簡易グラフの表示

```
1  #include <stdio.h>
2
3  int main(void) {
4      int i, j;
5      for (i = 1; i <= 10; i++) {
6          printf("%2d: ", i);
7          for (j = 0; j < i; j++) {
8              printf("*");
9          }
10         printf("\n");
11     }
12     return 0;
13 }
```

例 4-3 実行結果

```
$ ./a
1: *
2: **
3: ***
4: ****
5: *****
6: ****
7: *****
8: *****
9: *****
10: *****
```

1 から 10 までの数字を表示し、その数だけ右側に「*」を繰り返し表示するプログラムです。7 行目が、変数 j が 0 から i の間、処理を繰り返すという命令になっています。「*」を繰り返し表示する処理が終わったら、10 行目で改行を入れて、次の数字を表示する準備をします。このような入れ子構造は、二次元データや三次元データを扱う際に必ず必要になりますので、使いこなせるように練習しておきましょう。

演習 4-2 (ex4-2.c)

for 文の入れ子を利用して、九九の表を表示して下さい。

演習 4-2 表示例①

```

1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81

```

演習 4-2 表示例②

```

| 1  2  3  4  5  6  7  8  9
-----
1 | 1  2  3  4  5  6  7  8  9
2 | 2  4  6  8 10 12 14 16 18
3 | 3  6  9 12 15 18 21 24 27
4 | 4  8 12 16 20 24 28 32 36
5 | 5 10 15 20 25 30 35 40 45
6 | 6 12 18 24 30 36 42 48 54
7 | 7 14 21 28 35 42 49 56 63
8 | 8 16 24 32 40 48 56 64 72
9 | 9 18 27 36 45 54 63 72 81

```

4.2 while 文

繰り返し処理を行う命令として、for 文の他にも while 文があります。while 文は for 文と同じようにも使うことができますが、基本的には「繰り返し回数が決まった数でないとき」によく使われます。while 文の書式は以下に示すとおりです。

```
while ( 条件式 ) {  
    // 繰り返し実行する処理  
}
```

for 文と比べて、大分シンプルになっていることがわかると思います。while 文では、条件式を満たす間、波括弧で囲われた処理を繰り返し実行します。ただし、繰り返し実行する処理の部分が条件式と無関係である場合、永遠に繰り返されるので（無限ループ）注意が必要です。while 文の使用例として、次のソースコードを見てみましょう。

例 4-4 100 未満の 2 の累乗数を while 文で表示

```
1  #include <stdio.h>  
2  
3  int main(void) {  
4      int n = 1;  
5      while (n < 100) {  
6          printf("%d\n", n);  
7          n *= 2;  
8      }  
9      return 0;  
10 }
```

例 4-4 実行結果

```
$. /a  
1  
2  
4  
8  
16  
32  
64
```

4 行目： 整数型の変数 `n` を宣言し、1 を代入しています。

5 行目： `while` 文で、`n` が 1 より大きい間、処理を繰り返すことを表しています。`n` の値は 7 行目で 2 倍になっていきます。

このプログラムは、`n` の値を繰り返し 2 倍にしていき、100 を超えたら終了するというものです。実は、`for` 文を用いて次のように書き換えることも可能です。

例 4-5 100 未満の 2 の累乗数を `for` 文で表示

```

1 #include <stdio.h>
2
3 int main(void) {
4     int n;
5     for (n = 1; n < 100; n *= 2) {
6         printf("%d\n", n);
7     }
8     return 0;
9 }
```

どちらが読みやすいか、書きやすいかなどを考慮して、`for` 文と `while` 文を使い分けられるようになりましょう。

演習 4-3 (ex4-3.c)

整数値の入力を繰り返し受け取り、合計値を表示するプログラムを作成して下さい (`while` 文を使用)。また、合計値が 100 を超えたらプログラムを終了するようにして下さい。

※予期せぬ無限ループに陥ってしまった場合は、[Ctrl] + [C] で強制終了させることができます。

演習 4-3 実行例

\$./a

整数値を入力して下さい: 15

合計値: 15

整数値を入力して下さい: 60

合計値: 75

整数値を入力して下さい: 30

合計値: 105

合計値が 100 を超えたのでプログラムを終了します。

4.3 do～while 文

while 文とよく似た文に、do～while 文があります。while 文（および for 文）は、条件式を評価してから繰り返し処理の内容が実行されるのに対し、do～while 文では繰り返し処理の内容をまず実行してから、条件式を評価します。すなわち、while 文では処理の内容が 1 度も実行されない可能性があるのに対し、do～while 文であれば必ず 1 回は実行されます。do～while 文の書式は以下のとおりです。

```
do {  
    // 繰り返し実行する処理  
} while ( 条件式 );
```

do～while 文は、少なくとも 1 回は処理が実行されるという特徴から、入力を受け取り処理などで利用されることがあります。例えば、例 3-2 のソースコードを改良した以下のコードを見てみましょう。

例 4-6 while 文による例 3-2 の改良

```
1  #include <stdio.h>  
2  
3  int main(void) {  
4      int score;  
5  
6      do {  
7          printf("成績を入力して下さい (0~100): ");  
8          scanf("%d", &score);  
9      } while (score < 0 || 100 < score);  
10  
11     if (score >= 60) {  
12         printf("単位がもらえます。\\n");  
13     } else {  
14         printf("不可です。\\n");  
15     }  
16  
17     return 0;  
18 }
```

例 4-6 実行例

```
$ ./a
```

```
成績を入力して下さい (0～100) : -5
```

```
成績を入力して下さい (0～100) : 110
```

```
成績を入力して下さい (0～100) : 60
```

```
単位がもらえます。
```

7～8 行目で成績の入力を受け付け、9 行目の条件を満たす間、何度も繰り返し入力を促すソースコードとなっています。このループを抜け出すためには、score に 0 から 100 の数値を入れる必要があります。

演習 4-4 (ex4-4.c)

足し算の問題を表示して、正解が入力されるまで数値の入力を促すプログラムを作成して下さい (do～while 文を使用)。

※予期せぬ無限ループに陥ってしまった場合は、[Ctrl] + [C]で強制終了させることができます。

演習 4-4 実行例

```
$ ./a
```

```
9 + 19 = ?
```

```
解答を入力して下さい: 15
```

```
解答を入力して下さい: 30
```

```
解答を入力して下さい: 28
```

```
正解です。
```

4.4 break 文と continue 文

for 文と while 文による繰り返し処理をさらに使いこなすための、break 文と continue 文についても紹介します。これらは繰り返し処理の中で使われる文であり、それぞれ以下の役割を持ちます。

break 文・・・繰り返し処理を中断してループから抜ける

continue 文・・・繰り返し処理を中断してループの先頭に戻る

また、break 文と continue 文は、よく無限ループと組み合わせて利用されます。無限ループとはその名のとおり、処理を無限に繰り返すことを意味します。for 文では終了の条件式に何も書かないことで、while 文では条件式に真偽値の「真」を意味する“1”を入れることで、無限ループを作ることができます。

```
1 // for 文による無限ループ
2 for (;;) {
3     // 繰り返し実行する処理
4 }
5
6 // while 文による無限ループ
7 while (1) {
8     // 繰り返し実行する処理
9 }
```

※予期せぬ無限ループに陥ってしまった場合は、[Ctrl] + [C]で強制終了させることができます。

それでは実際に、無限ループと break 文、continue 文を組み合わせた例を見てみましょう。

例 4-7 break 文, continue 文による例 4-6 の改良

```
1 #include <stdio.h>
2
3 int main(void) {
4     int score;
5
6     while (1) {
7         printf("成績を入力して下さい (0~100): ");
8         scanf("%d", &score);
9         if (score < 0 || 100 < score) {
```

```

10         printf("0 から 100 の範囲で入力して下さい。\\n");
11         continue;
12     } else {
13         break;
14     }
15 }
16
17 if (score >= 60) {
18     printf("単位がもらえます。\\n");
19 } else {
20     printf("不可です。\\n");
21 }
22 return 0;
23 }

```

例 4-7 実行例

```

$ ./a
成績を入力して下さい (0~100) : -5
0 から 100 の範囲で入力して下さい。
成績を入力して下さい (0~100) : 110
0 から 100 の範囲で入力して下さい。
成績を入力して下さい (0~100) : 60
単位がもらえます。

```

例 4-6 では、入力された数値が 0~100 の範囲外であった場合、何も表示せずに再び入力を促すだけでした。一方、例 4-7 では 10 行目で「0 から 100 の範囲で入力して下さい。」と表示した後に continue 文を用いることで、よりユーザーにわかりやすいプログラムとなっています。また、範囲が正しかった場合は 13 行目の break 文で、while 文の無限ループから抜け出しています。

4.5 課題

課題 4-1

正の整数 n を受け取り、for 文を使って $n! (= 1 \times 2 \times \dots \times n)$ の値を表示するプログラムを作成して下さい。

課題 4-2

for 文を使って円周率を求めてみましょう。計算には、ライプニッツの公式として知られる以下の級数を用いて下さい。また、繰り返しの回数を 100 回、100 万回、10 億回と変化させて、実際の円周率との誤差を確認して下さい。

$$\pi = 4 \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + (-1)^{n-1} \frac{1}{2n-1} + \dots \right)$$

ヒント 1：奇数項と偶数項で符号が反転しています。

ヒント 2：結果の表示が省略されないよう「%.15lf」等としましょう。

課題 4-3

課題 4-2 のプログラムを、while 文を使って書き直して下さい。ただし終了条件には、誤差がある値（例えば 1×10^{-9} ）よりも小さくなった場合を設定して下さい。

補足：math.h の利用

本章の課題を効率よく解くために、math.h を紹介します（詳細は付録にて）。math.h とは、さまざまな数学の関数や定数が用意されたヘッダファイルであり、利用する場合は「#include <math.h>」のようにして読み込みます。例えば、math.h の中には円周率が定数（マクロ）として定義されており、「M_PI」と入力することで円周率の値を利用することができます。また、fabs()関数を用いることで、数値の絶対値を計算することができます。

math.h の使用例

```

1 #include <stdio.h>
2 #include <math.h>
3
4 int main(void) {
5     printf(" 3.1 - pi  = %.15lf\n", 3.1 - M_PI);           // 差の計算
6     printf("|3.1 - pi| = %.15lf\n", fabs(3.1 - M_PI));      // 差の絶対値の計算
7     return 0;
8 }
```