第4章 「プログラムの流れの繰返し」のまとめ

4.1 用語のまとめ

do ～ while 文 (教 p.74)

```
do 文1 while ( 式1 ) ;
```

まず、文$_1$（ループ本体と呼ばれる）を実行する。式$_1$ が＿＿（真）である限り、＿＿の実行を繰り返す。

注: 必ず 1 回はループ本体を実行する。

あとで紹介する while 文や for 文に比べると、do ～ while 文の実際のプログラムでの**使用頻度は低い。**

プログラムの実行が止まらなくなったときは、Ctrl-c で強制終了する。

複合文（ブロック）内での宣言 (教 p.75)

（do ～ while 文に限らず）ブロックの中で宣言された変数は、そのブロックでのみ有効である。

論理否定演算子 (教 p.77)

単項演算子の「＿」は、真偽を逆にする演算子で、論理否定演算子とも言う。

複合代入演算子 (教 p.80)

「*=」, 「/=」, 「%=」, 「+=」, 「-=」, などのことである。例えば、 `sum += t` は＿＿＿＿＿＿＿＿とほぼ同じ意味になる。つまり、この代入を実行した後の `sum` の値は、実行する前の `sum` の値に `t` を加えた値になる。

後置増分演算子・前置増分演算子 (教 p.81) (教 p.88)

a++	a の値を一つだけ増やす	（式全体の値は、＿＿＿＿＿＿の値）
a--	aの値を一つだけ減らす	（式全体の値は、＿＿＿＿＿＿の値）
++a	aの値を一つだけ増やす	（式全体の値は、＿＿＿＿＿＿の値）
--a	aの値を一つだけ減らす	（式全体の値は、＿＿＿＿＿＿の値）

Q 4.1.1 次のプログラムの断片の出力は何か? 答 ＿＿＿＿

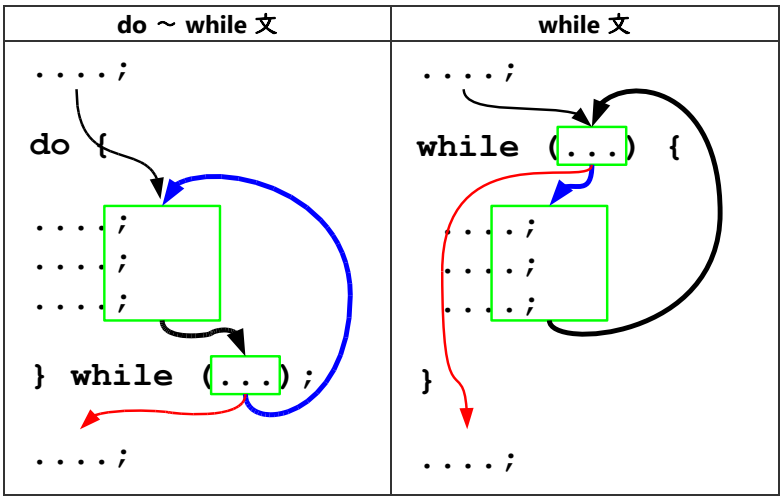
```
1  n = 3;
2  do {
3      printf("%d ", n);
4  } while (n-- > 0);
```

while 文 (教 p.82)

```
while ( 式1 ) 文1
```

式$_1$ が＿＿（偽）でない限り、＿＿（ループ本体）の実行を繰り返す。

注: ループ本体が一度も実行されないことがある。



Q 4.1.2 次のプログラムの断片の出力は何か? 答 ＿＿＿

```
1  n = 3;
2  while (n-- > 0) {
3      printf("%d ", n);
4  }
```

文字定数 (教 p.86)

1 文字を＿＿＿＿「'」～「'」で囲んだもののことである。「\n」や「\t」などの拡張表記は 1 文字として扱われる。

putchar 関数 (教 p.87)

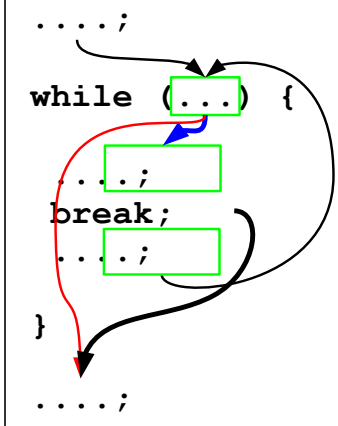
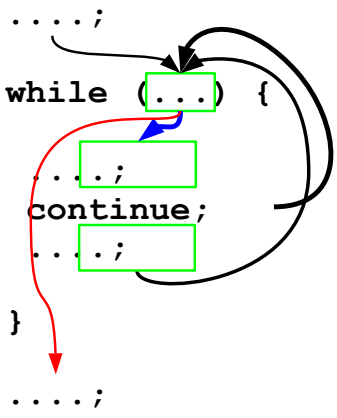
引数として受け取った文字を標準出力に出力する。

break 文 (教 p.92)

（もっとも内側の）繰返し文（do ～ while 文, while 文, for 文）を＿＿＿＿。
（外側の繰返し文を一気に抜け出ることはできない。）

continue文（参考） (教 p.93)

（もっとも内側の）繰返し文のはじめ（do ～ while文、 while 文の場合は条件式、 for 文の場合は第 3 式）にもどる。

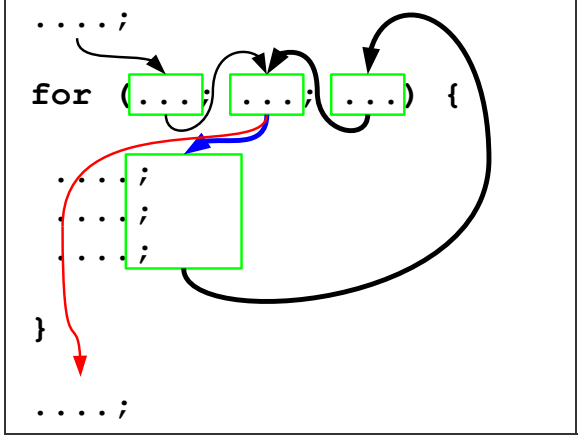
break 文	continue 文
	

for 文 (教 p.94)

```
for ( 式1; 式2; 式3 ) 文1
for ( 宣言1 式2; 式3 ) 文1
```

ループに入る前にまず＿＿（または ＿＿＿）を実行する。
＿＿が非0（真）である間、＿＿（ループ本体）と ＿＿を繰返し実行する。

詳細: 式$_1$～式$_3$は省略可能である。 式$_2$を省略したときは、1 (真) と書くのと同じ意味になる。

for 文


左のような for 文は右に示す while 文と（ほぼ）等価である。

for 文	（ほぼ）等価な while 文

for (*A* ; *B* ; *C*) {
　ループ本体
}
	＿＿＿;
	while (＿＿＿) {
	＿＿＿＿＿＿＿＿＿＿＿＿＿＿
	＿＿＿＿ ;
	}

では、なぜ左の書き方が好まれるか？ — 繰り返しを制御する変数に対する処理が、一箇所にまとまっていて、一目でどのような繰り返しか理解しやすいから

である。

一定回数の繰返し (教 p.97)

for 文には、良く使う決まり文句的な形がある。

① `for (i = 0; i < n; i++) …` i が _____ n 回繰り返す
② `for (i = 1; i <= n; i++)…` i が _____ n 回繰り返す
③ `for (i = n; i > 0; i--)…` i が _____ n 回繰り返す
④ `for (i = n - 1; i >= 0; i--)…` i が _____ n 回繰り返す

Q 4.1.3 上記の形で、`n` が 0 や負の数だった場合はどうなるか？

_____

Q 4.1.4 上記のそれぞれの形でループを抜けたあとの `i` の値は何か？（ただし、`n` は非負とする。）

① ____ ② ____ ③ ____ ④ ____

空文 (教 p.101)

文 (statement) に以下を追加する。

分類	一般形	補足説明
空文	;	“何もしない”文、`{}` と書いても同じ。

多重ループ (教 p.102)

for 文や while 文などのループ本体が、また for 文や while 文などの繰返し文を含んでいることを二重ループという。二重・三重・… ループをまとめて、多重ループという。特別な文法や実行規則があるわけではない。

コンマ演算子 (教 p.233)

式$_1$, *式*$_2$

という**式**は、*式*$_1$、*式*$_2$をこの順に評価し、 ____ の値を**捨て**、 ____ の値（と型）を持つ。

注: 関数を呼び出すときに引数を区切るコンマ（例：`printf("%d %d", i, j)`）はコンマ演算子**ではない**。

コンマ演算子の例 (comma.c)

```c
#include <stdio.h>

int main(void) {
    int i, j;
    for (i = 0, j = 6; i < j; i++, j--) {
        printf("i = %d, j = %d\n", i, j);
    }
    return 0;
}
```

Q 4.1.5 このプログラムの出力はどうなるか？

キーワード (教 p.108)

`if` や `else` など C 言語にとって特別な意味のある単語を _____(keyword) と呼ぶ。変数名などに使用することはできない。(ただし、変数名などの一部に使用するのは構わない。)

自由形式 (教 p.110)

C 言語では、原則としてレイアウト（空白の数や改行）はプログラム の意味に影響を及ぼさない。空白がいくつ連続しても空白 1 文字と同じであり、改行も空白と同じである。

注意すべきところ:

- 前処理指令 (`#include` ..., `#define` ...) などの途中では、改行できない。
- 文字列リテラル、文字定数の途中でも、改行できない。

4.2 プログラム例

整数値を逆順に (reverse.c) (教科書 List 4-10 に類似)

```
 1 #include <stdio.h>
 2
 3 int main(void) {
 4     int no = 12345;
 5
 6     do {
 7         printf("%d", no % 10);
 8         no /= 10;
 9     } while (no > 0);
10
11     return 0;
12 }
```

Q 4.2.1 `no` の値の変化はどうなるか？

	1 回目	2 回目	3 回目	4 回目	5 回目
7 行	_____	_____	_____	_____	_____
9 行	_____	_____	_____	_____	_____

Q 4.2.2 6 ～ 9 行めの do ～ while 文の代わりに以下のような while 文を使うと、振舞いがどう変わるか？

```
1     while (no > 0) {
2         printf("%d", no % 10);
3         no /= 10;
4     }
```

典型的な for 文（階乗の計算）(fact.c)

```c
#include <stdio.h>

int main(void)  {
    int i, n, fact = 1;

    printf("正の数を入力してください。 ");
    scanf("%d", &n);
    for (i = 1;  /* 通常 1 行に書くところを 3 行にわけた。 */
                i <= n;
                        i++) {
        fact = fact * i;
    }
    printf("あなたの入力した数の階乗は %dです。\n", fact);

    return 0;
}
```

Q 4.2.3 i, fact の値の変化（n が 5 のとき）はどうなるか？

行	1 回目		2 回目		3 回目		4 回目		5 回目	
	i	fact	i	fact	i	fact	i	fact	i	fact
9										
11		→		→		→		→		→
10	→		→		→		→		→	

典型的な for 文（正 n 角形の座標の出力）(polygon.c)

```c
#include <stdio.h>
#include <math.h> /* sin, cos のために必要 — 教 p.217*/

int main(void) {
    int n, i;

    printf("nを入力して下さい: "); scanf("%d", &n);
    for(i = 0; i < n; i++) {
        double theta1 = 2 * 3.1416 *  i / n;
        double theta2 = 2 * 3.1416 *  (i + 1) / n;
        printf("%.3f %.3f %.3f %.3f\n",
               100 * cos(theta1), 100 * sin(theta1),
               100 * cos(theta2), 100 * sin(theta2));
    }

    return 0;
}
```

二重ループ（数の三角形）(triangle.c)

```c
#include <stdio.h>

int main(void) {
```

```c
    int i, j, n;
    printf("n を入力して下さい: "); scanf("%d", &n);
    for (i = 1/*①*/; i <= n/*②*/; i++/*③*/) {
        for (j = 1/*④*/; j <= i/*⑤*/; j++/*⑥*/) {
            printf("%d", j % 10);/*⑦*/
        }
        printf("\n");/*⑧*/
    }
    return 0;
}
```

```
n を入力して下さい: 4↵
1
12
123
1234
```

Q 4.2.4 n = 3 のとき、ループ内の式、文はどの順で実行されるか？

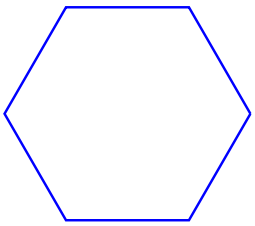
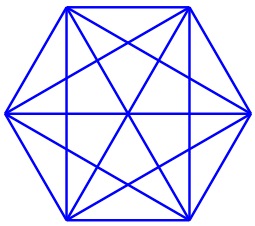
二重ループ（ダイアモンド図形の座標の出力）(diamond.c)

```c
#include <stdio.h>
#include <math.h> /* sin, cos のために必要 – 教 p.217 */

int main(void) {
    int n, i, j;

    printf("nを入力して下さい: "); scanf("%d", &n);
    for (i = 0/*①*/; i < n - 1/*②*/; i++/*③*/) {
        double theta1 = 2 * 3.1416 *  i / n/*④*/;
        for (j = i + 1/*⑤*//* ! */; j < n/*⑥*/;
j++/*⑦*/) {
            double theta2 = 2 * 3.1416 *  j / n/*⑧*/;
            printf("%.3f %.3f %.3f %.3f\n",
                   100 * cos(theta1), 100 *
sin(theta1),
                   100 * cos(theta2), 100 *
sin(theta2));
            /*⑨*/
        }
    }

    return 0;
}
```

正 n 角形（n = 6）	ダイアモンド図形（n = 6）
	

Q 4.2.5 `n = 4` のとき、ループ内の式、文はどの順で実行されるか？

4.3 文法のまとめ

文 (statement)

に以下を追加する。

分類	一般形	補足説明
do ～ while 文	`do` 文 `while （` 式 `）;`	（教 p.74）
while 文	`while （` 式 `）` 文	（教 p.82）
continue 文	`continue ;`	（教 p.93）
for 文	`for （` 式 `;` 式 `;` 式 `）` 文	（教 p.94）

式 (expression)

に以下を追加する。

分類	一般形	補足説明
後置演算	式 後置演算子	C の後置演算子は ++, -- のみ（教 p.81）
コンマ演算子	式, 式	（教 p.233）