

プログラミング演習I

繰り返し

do 文(List4-1)

List 4-1

chap04/list0401.c

```
/*
 * 読み込んだ整数値は奇数であるか偶数であるか（好きなだけ繰り返せる）
 */

#include <stdio.h>

int main(void)
{
    int retry;          /* 処理を続けるか */
    do {
        int no;
        printf("整数を入力してください：");
        scanf("%d", &no);

        if (no % 2)
            puts("その数は奇数です。");
        else
            puts("その数は偶数です。");

        printf("もう一度？【Yes...0/No...9】：");
        scanf("%d", &retry);
    } while (retry == 0);

    return 0;
}
```

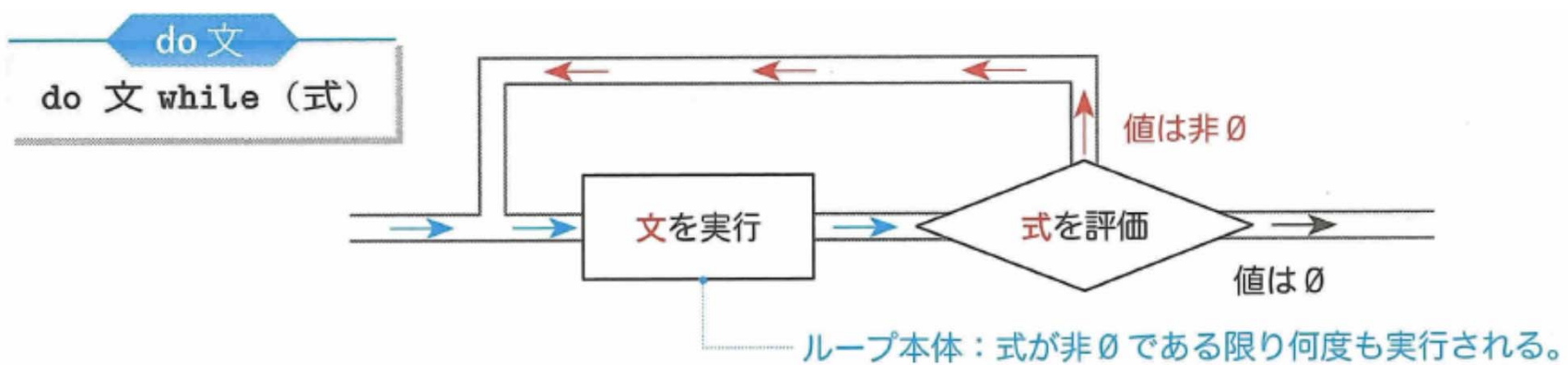
複合文内で宣言した変数は、
複合文内のみで有効

実行例

```
整数を入力してください：17
その数は奇数です。
もう一度？【Yes...0/No...9】：0
整数を入力してください：36
その数は偶数です。
もう一度？【Yes...0/No...9】：9
```

List 3-4 と同じ

()内为非ゼロ(真)である間は、do {}内の
複合文を繰り返す



● Fig.4-2 do 文のプログラムの流れ

一定範囲の値の読み込み

List 4-2

chap04/list0402.c

```
/*
 * 読み込んだ整数値に応じてジャンケンの手を表示 (0, 1, 2のみを受け付ける)
 */

#include <stdio.h>

int main(void)
{
    int hand;    /* 手 */

    do {
        printf("手を選んでください【0…グー／1…チョキ／2…パー】 : ");
        scanf("%d", &hand);
    } while (hand < 0 || hand > 2);

    printf("あなたは");
    switch (hand) {
        case 0: printf("グー");    break;
        case 1: printf("チョキ");  break;
        case 2: printf("パー");    break;
    }
    printf("を選びました。 \n");

    return 0;
}
```

実行例

```
手を選んでください【0…グー／1…チョキ／2…パー】 : 3
手を選んでください【0…グー／1…チョキ／2…パー】 : -2
手を選んでください【0…グー／1…チョキ／2…パー】 : 1
あなたはチョキを選びました。
```

chap04/list0402a.c

別解

```
!(hand >= 0 && hand <= 2)
```

handの値は0, 1, 2のいずれかとなる。

do 文の繰返しの条件は、『**hand**が**妥当な値** (0 以上かつ 2 以下) でなければ…」と表現したほうが、しっくりします。それを表すのが、左ページの《別解》として示した式です。do 文の**制御式**を**別解**に置きかえても、プログラムは同じ動作をします。

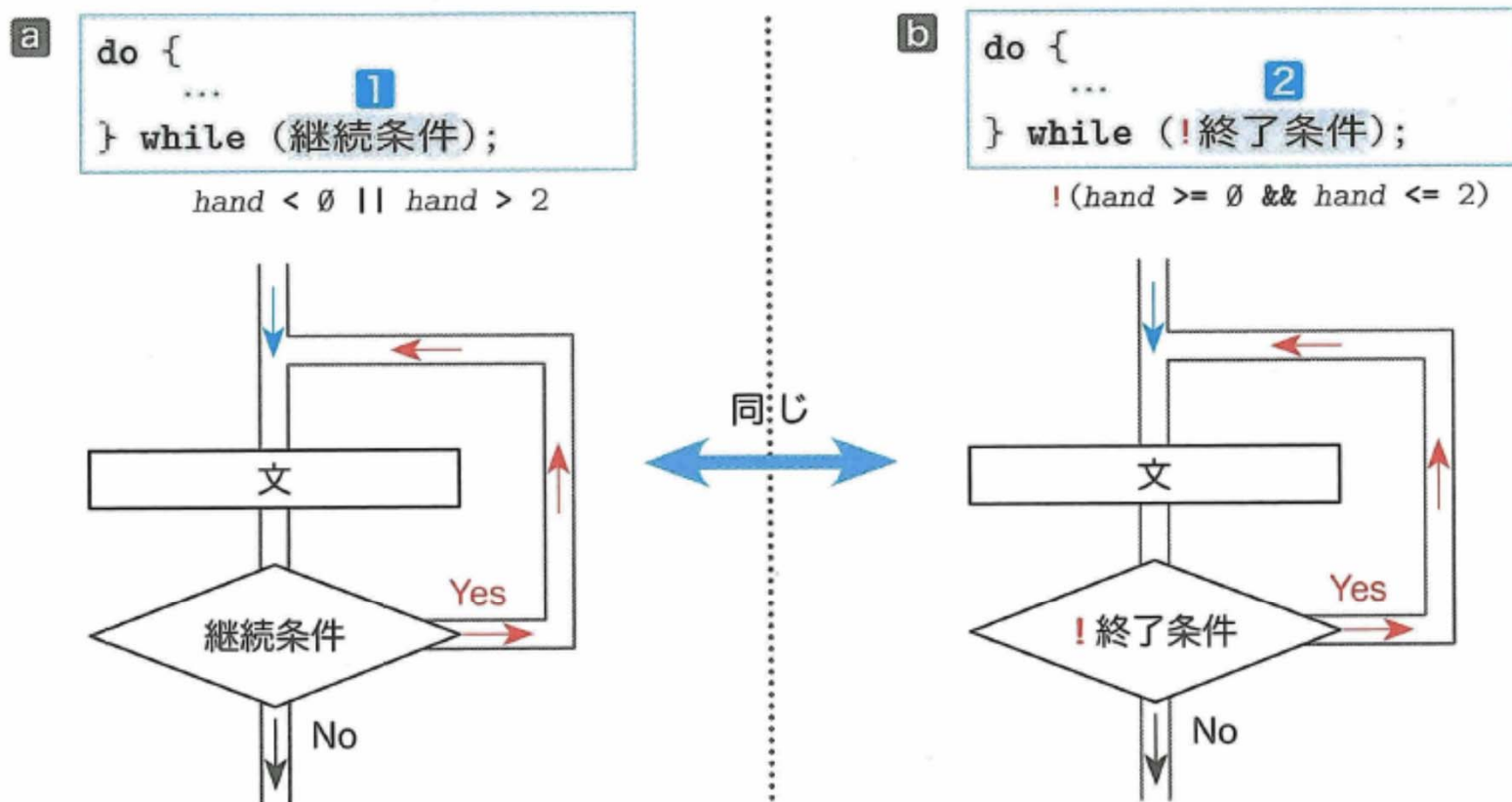
ド・モルガンの法則

『各条件の否定をとって、論理積・論理和を入れかえた式』の否定が、もとの条件と同じになることは、**ド・モルガンの法則** (De Morgan's theorem) と呼ばれます。この法則を一般的に示すと、以下のようになります。

- $x \ \&\& \ y$ と $!(!x \ || \ !y)$ は等しい。
- $x \ || \ y$ と $!(!x \ \&\& \ !y)$ は等しい。

Fig.4-4 a)に示すように、式**1**は繰返しを続けるための**継続条件**です。

一方、論理否定演算子!を使って書きかえた式**2**は、図b)に示すように、繰返しを終了するための**終了条件の否定**です。



● Fig.4-4 do 文の継続条件と終了条件


```
/*
 *  整数値を次々と読み込んで合計と平均を表示
 */
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
1 int sum = 0;    /* 合計 */
  int cnt = 0;    /* 整数値の個数 */
  int retry;      /* 処理を続けるか */
```

```
  do {
```

```
    int t;
```

```
    printf("整数値を入力してください：");
    scanf("%d", &t);
```

```
2 sum = sum + t;    /* sumにtを加えた値をsumに代入 (sumにtを加える) */
  cnt = cnt + 1;    /* cntに1を加えた値をcntに代入 (cntに1を加える) */
```

```
    printf("まだ？<Yes...0/No...9>：");
    scanf("%d", &retry);
```

```
  } while (retry == 0);
```

```
  printf("合計は%dで平均は%.2fです。\\n", sum, (double)sum / cnt);
```

```
  return 0;
```

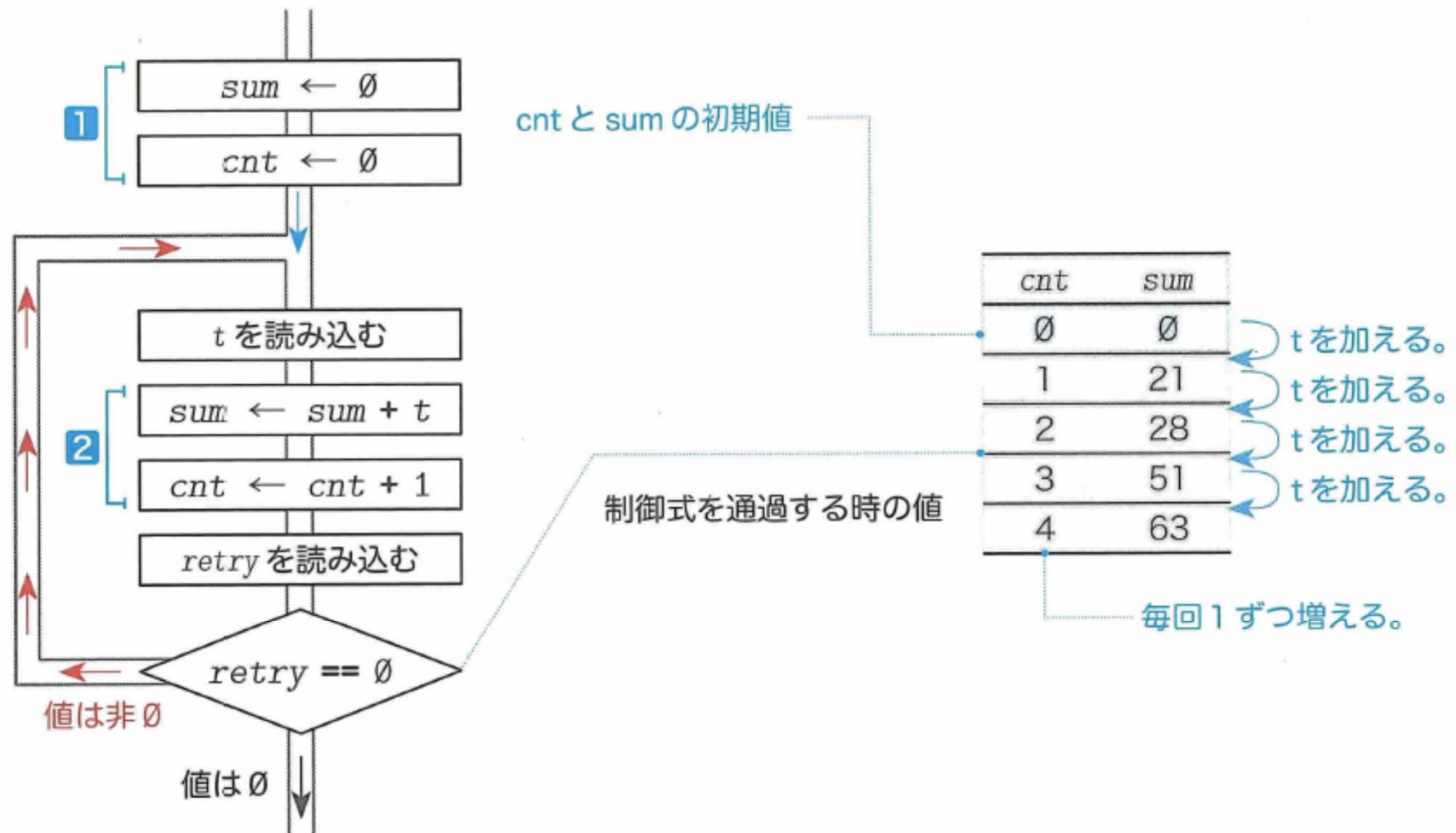
```
}
```

実行例

```
整数値を入力してください： 21
まだ？<Yes...0/No...9>： 0
整数値を入力してください： 7
まだ？<Yes...0/No...9>： 0
整数値を入力してください： 23
まだ？<Yes...0/No...9>： 0
整数値を入力してください： 12
まだ？<Yes...0/No...9>： 9
合計は63で平均は15.75です。
```

小数部を2桁表示。

キャスト式 (p.35 で学習)。



● Fig.4-5 合計を求めていくプログラムの流れ

いろいろな省略記法

- 複合代入演算子 (p.78)
 - $a+=\text{変数(数字)}$ \Rightarrow $a=a+\text{変数(数字)}$ (例 $a+=2$)
 - 四則演算子を含む2項演算子すべてが可能($/, \%, +, -, <<, >>, \&, ^, |$) (まだ出てきていない演算子もある)。
 - ($a/=2$ は $a=a/2$, $a*=2$ は $a=a*2$)
 - よく使うのは $+$ 、 $-$ だけ。あまり使うと可読性が悪くなる。
- 後置増分(減分)演算子 (p.79)
 - $a++$ ($a--$): 変数 a を参照し、式全体を評価した後、変数 a に1を足す(引く)
- 前置増分(減分)演算子 (p.86)
 - $++a$ ($--a$): 変数 a を参照する前に、変数 a に、1を足す(引く)

```
/*
 *  整数値を次々と読み込んで合計と平均を表示
 */
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
1 int sum = 0;      /* 合計 */
  int cnt = 0;      /* 整数値の個数 */
  int retry;        /* 処理を続けるか */
```

```
do {
```

```
    int t;
```

```
    printf("整数値を入力してください：");
    scanf("%d", &t);
```

@List4-4

2

```
    sum+=t;
    cnt++;
```

```
    t; /* sumにtを加えた値をsumに代入 (sumにtを加える) */
```

```
    1; /* cntに1を加えた値をcntに代入 (cntに1を加える) */
```

```
    printf("まだ？<Yes...0/No...9>：");
    scanf("%d", &retry);
```

```
} while (retry == 0);
```

```
printf("合計は%dで平均は%.2fです。\\n", sum, (double)sum / cnt);
```

```
return 0;
```

```
}
```

小数部を2桁表示。

キャスト式 (p.35 で学習)。

実行例

```
整数値を入力してください： 21
まだ？<Yes...0/No...9>： 0
整数値を入力してください： 7
まだ？<Yes...0/No...9>： 0
整数値を入力してください： 23
まだ？<Yes...0/No...9>： 0
整数値を入力してください： 12
まだ？<Yes...0/No...9>： 9
合計は63で平均は15.75です。
```

● **Table 4-3** 後置増分演算子と後置減分演算子

後置増分演算子	<code>a++</code>	<code>a</code> の値を一つだけ増やす（式全体を評価すると、増分前の値となる）。
後置減分演算子	<code>a--</code>	<code>a</code> の値を一つだけ減らす（式全体を評価すると、減分前の値となる）。

ほぼ同じ。

→ `a = a + 1` `/* aに1を加えた値をaに代入 */`

→ `a++` `/* aをインクリメントする（aに1を加える） */`

ほぼ同じ。

→ `a = a - 1` `/* aから1を減じた値をaに代入 */`

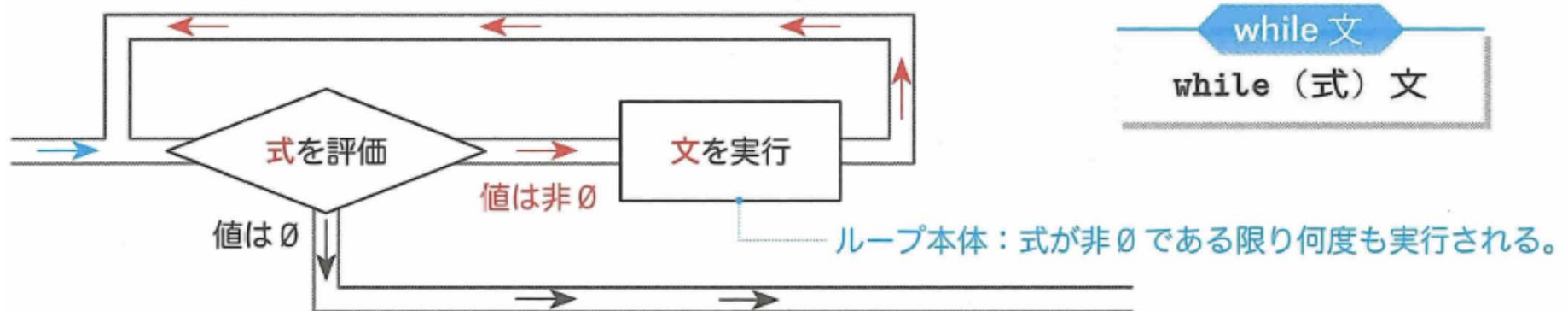
→ `a--` `/* aをデクリメントする（aから1を減じる） */`

● **Fig.4-7** 後置増分演算子と後置減分演算子

while文



● Fig.4-8 while 文の構文図



● Fig.4-9 while 文のプログラムの流れ

```
/* 読み込んだ整数値を0までカウントダウン */
#include <stdio.h>
int main(void)
{
    int no;

    printf("正の整数を入力してください：");
    scanf("%d", &no);

    while (no >= 0) {
        printf("%d.", no);
        no--; /* noの値をデクリメント */
    }
    printf("\n"); /* 改行 */

    return 0;
}
```

この部分が繰り返
文(while文)

実行例 1

正の整数を入力してください：5
5 4 3 2 1 0

実行例 2

正の整数を入力してください：0
0

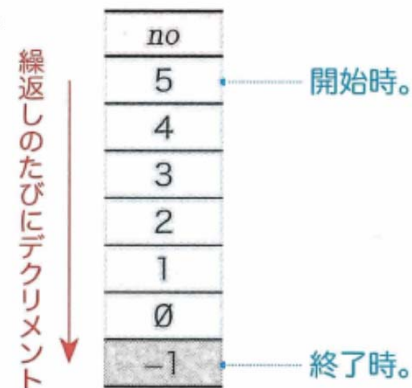
正でなく0が入力されても表示される。

実行例 3

正の整数を入力してください：-5

改行文字だけが出力される。

制御式 $no \geq 0$ の通過時の値。



● Fig.4-10 noの値の変化


```

/*
 * 読み込んだ整数値を0までカウントダウン
 */
#include <stdio.h>

int main(void)
{
    int no;

    printf("正の整数を入力してください：");
    scanf("%d", &no);

    while (no >= 0) {
        printf("%d ", no);
        no--;
    }

    printf("\n");
    return 0;
}

```

この部分が繰り返
文(while文)

@List4-6

while(no >= 0)
printf("%d ", no--);

スペース

/* 改行 */

後置増分演算子と後置減分演算子を紹介した Table 4-3 (p.79) を、もう一度読み直

てみましょう。a-- の説明は、以下のようになっています。

a の値を一つだけ減らす (式全体を評価すると、減分前の値となる)。

したがって、printf 関数を呼び出して no-- を
表示する際は、

- ① no の値を表示する。
- ② no の値をデクリメントする。

という2段階の手順が踏まれます。

すなわち、『no の値を表示した直後にデクリメ
ントする。』というわけです。

実行例 1

正の整数を入力してください：5
5 4 3 2 1 0

実行例 2

正の整数を入力してください：0
0

でなく0が入力されても表示される。

実行例 3

正の整数を入力してください：-5

評価するとデクリメント前の値が得られる。



※ no が 11 であるとする。

11 が得られた後にデクリメントする。

● Fig.4-11 後置減分演算式の評価


```
/*
 * 読み込んだ正の整数値までカウントアップ
 */
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("正の整数を入力してください：");
    scanf("%d", &no);

    i = 0;
    while (i <= no)
        printf("%d ", i++);    /* iの値を表示した後にインクリメント */
    printf("\n");              /* 改行 */

    return 0;
}
```

実行例

正の整数を入力してください：12

0 1 2 3 4 5 6 7 8 9 10 11 12

一定回数の繰返し(while文で)

List 4-8

chap04/list0408.c

```
/*
 * 読み込んだ整数の個数だけ * を連続表示
 */
#include <stdio.h>

int main(void)
{
    int no;

    printf("正の整数: ");
    scanf("%d", &no);

    while (no-- > 0)
        putchar('*');
    putchar('\n');

    return 0;
}
```

実行例 1

正の整数: 15

実行例 2

正の整数: 0

実行例 3

正の整数: -5

文字と文字列

- 今まで、printf(“複数文字”)という表記法を使ってきた。
- 実は“複数文字”(ダブルクオートで括られた0文字以上の文字の並び)を文字列と呼ぶ。
- また、1文字だけを扱いたい場合は、' 1文字' のようにシングルクオートで囲む
- 実は
 - “文字列”は、計算機の中では、一文字ごとの文字コードの列の最後に¥0 (NULL文字、数字の0と等価)が付加されている。そうしないと、あらかじめ長さのわからない文字列を扱うことができない。
 - 一方、1文字だけの' 1文字' は、単に参照時に文字コードに変換されるだけ

文字コード表 (ASCIIコード)

							b7	0	0	0	0	1	1	1	1
							b6	0	0	1	1	0	0	1	1
							b5	0	1	0	1	0	1	0	1
b7	b6	b5	b4	b3	b2	b1		0	1	2	3	4	5	6	7
			0	0	0	0	0 (0)	NUL	DLE	SP	0	@	P	'	p
			0	0	0	1	1 (1)	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2 (2)	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3 (3)	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4 (4)	EOT	DC4	\$	4	D	T	d	t
			0	1	0	1	5 (5)	ENQ	NAC	%	5	E	U	e	u
			0	1	1	0	6 (6)	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7 (7)	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8 (8)	BS	CAN	(8	H	X	h	x
			1	0	0	1	9 (9)	HT	EM)	9	I	Y	i	y
			1	0	1	0	10 (A)	LF/NL	SUB	*	:	J	Z	j	z
			1	0	1	1	11 (B)	VT	ESC	+	;	K	[k	{
			1	1	0	0	12 (C)	FF	FS	,	<	L	\	l	
			1	1	0	1	13 (D)	CR	GS	-	=	M]	m	}
			1	1	1	0	14 (E)	SO	RS	.	>	N	^	n	~
			1	1	1	1	15 (F)	SI	US	/	?	O	_	o	DEL

↑()は16進数。 *JISでは、\は¥、~は-となります。

ちょっと余談

- ASCIIコード表を見ると、'A'は16進数で41、'B'は42。C言語では'B'-'A'なんて演算もできる。
- 2進数については、ちゃんと復習しておくこと。
- 2進数の4桁を一まとまりにしたのが16進数
 - 例えば00010001(2進数) => 11(16進数)
 - 01111001(2進数) => 79
 - 11111111(2進数) => FF
 - 10=>A, 11=>B, 12=>C, 13=>D, 14=>E, 15=>Fと表現する。

もうひとつ余談

- puts(“文字列”) は put strings(文字列)の意味
- putchar(‘文字’)は put character (文字)の意味
- printf(“書式”,式)は print format(書式)の意味
 - formatするには時間がかかるので、単純な文字列を出力するにはputsのほうが圧倒的に早い

do文とwhile文の違い

重要 do 文のループ本体は少なくとも1回は実行されるのに対し、while 文のループ本体は1回も実行されない可能性がある。

繰返しの継続条件の判定のタイミングが、do 文と while 文とでまったく異なります。

- a** do 文 … **後判定繰返し**：ループ本体を実行した後に判定を行う。
- b** while 文… **前判定繰返し**：ループ本体を実行する前に判定を行う。

▶ 次節で学習する for 文は、前判定繰返しです。

前置増分演算子と前置減分演算子

List 4-9

chap04/list0409.c

```
/*
 指示された個数だけ整数を読み込んで合計値と平均値を表示
*/
```

```
#include <stdio.h>
```

```
int main(void)
{
```

```
    int i = 0;
```

```
    int sum = 0;
```

```
    int num, tmp;
```

/* 合計値 */

```
    printf("整数は何個ですか：");
```

```
    scanf("%d", &num);
```

```
    while (i < num) {
```

```
        printf("No.%d：", ++i);
```

```
        scanf("%d", &tmp);
```

```
        sum += tmp;
```

```
    }
```

```
    printf("合計値：%d\n", sum);
```

```
    printf("平均値：%.2f\n", (double)sum / num);
```

```
    return 0;
```

```
}
```

宣言と同時に初期化

実行例

整数は何個ですか：6

No.1：65

No.2：23

No.3：47

No.4：9

No.5：153

No.6：777

合計値：1074

平均値：179.00

/* iの値をインクリメントした後に表示 */

i=0;

while (i < num) {

文;

i++; (この場合は++iでも同じ)

}

は、文をnum回繰り返すときの決まり構文

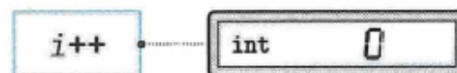
a 前置増分演算式

インクリメント後の値が得られる。



b 後置増分演算式

インクリメント前の値が得られる。



※いずれも `i` が `0` であるとする。

● **Fig.4-12** 増分演算式の評価

前置増分演算子は、インクリメントを行うという点では、後置増分演算子と同じですが、インクリメントのタイミングが異なります。それを対比して図示したのが、**Fig.4-12** です。

網かけ部で `++i` を表示する際は、以下の2段階の手順が踏まれます。

- ① `i` の値をインクリメントする。
- ② `i` の値を表示する。

すなわち、『`i` の値を表示する直前にインクリメントする。』というわけです。そのため、最初に表示される `i` の値は、`0` をインクリメントした後の `1` となります。

FOR文

- プログラムの大部分は繰り返し部分である。
→ WHILE文を使って、実現可能
- でも、明示的に“回数を繰り返すこと”を意図するために、FOR文がある。

List 4-11

chap04/list0411.c

```
/*
 * 読み込んだ正の整数値までカウントアップ (for文)
 */
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("正の整数を入力してください：");
    scanf("%d", &no);

    for (i = 0; i <= no; i++)
        printf("%d ", i);
    putchar('\n'); /* 改行 */

    return 0;
}
```

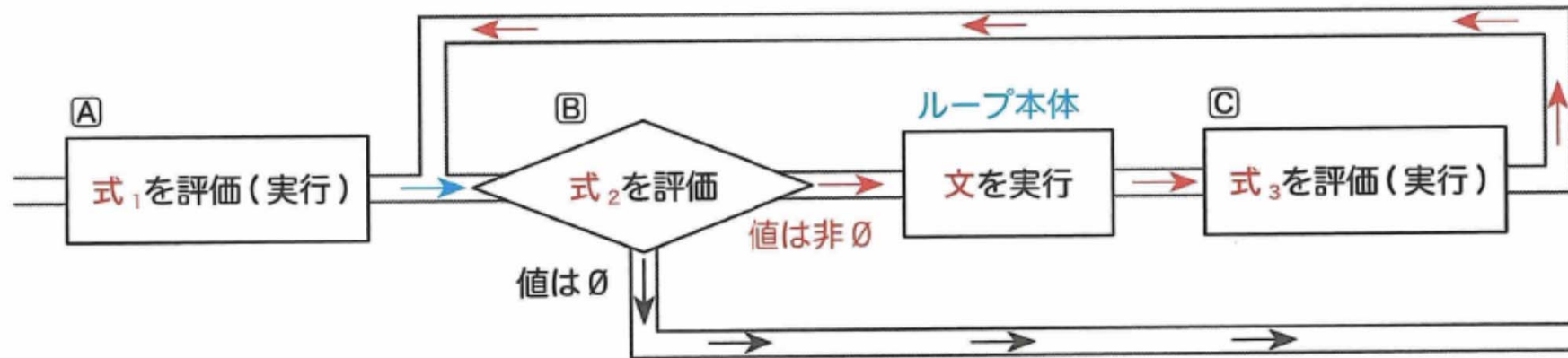
実行例

正の整数を入力してください：12 
0 1 2 3 4 5 6 7 8 9 10 11 12

```
/*--- 参考：List 4-7 ---*/
i = 0;
while (i <= no)
    printf("%d ", i++);
printf("\n");
```

for 文

for (式₁; 式₂; 式₃) 文

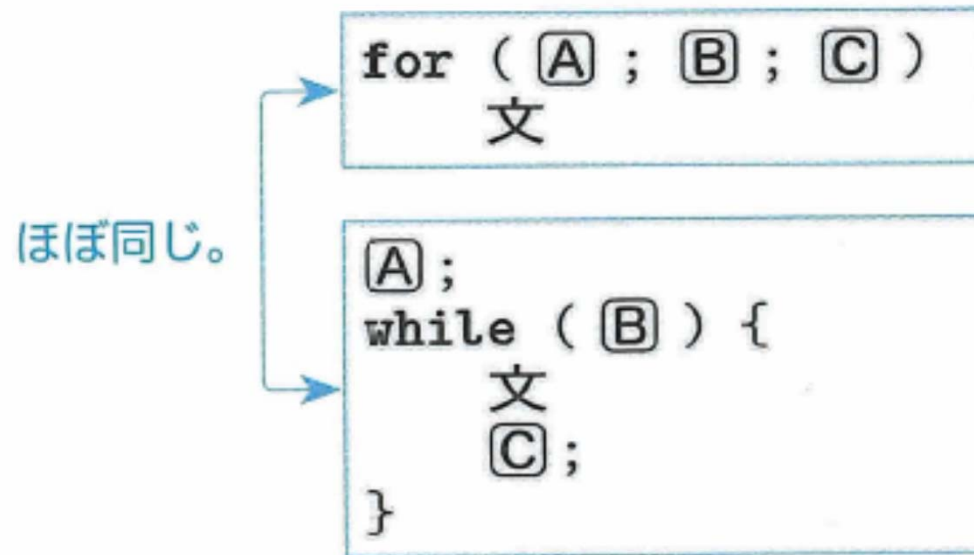


● Fig.4-16 for 文のプログラムの流れ

- ① 《前処理》として、A部を評価・実行する。
- ② 《継続条件》であるB部の制御式が非0であれば、文（ループ本体）を実行する。
- ③ 文の実行後に、《後始末的な処理》または《次の繰返しのための準備》であるC部を評価・実行して、②に戻る。

式の部分は、で区切り複数文がかける。
例 for(i=0,j=0; i<10; i++,j++)

FOR文=WHILE文



● Fig.4-17 for 文と while 文

A(前処理部) この部分はループの前1度しか実行されない

B(制御式) 繰り返しを行うかどうかの判定を行う制御式がはいる。この式を評価した結果非0の場合はループ本体が実行される。またB部を省略(空白)した場合は、無限ループとなる。

(for (;;) 文 => 文を無限回実行

C(後処理部) ループの最後に実行される文

典型的な利用法

List 4-12

chap04/list0412.c

```
/*
 * 読み込んだ整数の個数だけ*を連続表示 (for文)
 */

#include <stdio.h>

int main(void)
{
    int i, no;

    printf("正の整数: ");
    scanf("%d", &no);

    for (i = 1; i <= no; i++)
        putchar('*');
    putchar('\n');

    return 0;
}
```

実行例

正の整数: 15

```
/*--- 参考: List 4-8 ---*/
while (no-- > 0)
    putchar('*');
putchar('\n');
```

(A部) iを1で初期化し

(B部) iがnoより小さいか等しい間、ループ内を繰り返す

(C部) ループ内の文が終了後iに1を足す

いろいろなループ構成法

```
for (i = 0; i < n; i++)  
  文
```

繰返し終了時の*i*の値は*n*。
*n*の値は変化しない。

```
while (n-- > 0)  
  文
```

繰返し終了時の*n*の値は-1。

```
for (i = 1; i <= n; i++)  
  文
```

繰返し終了時の*i*の値は*n* + 1。
*n*の値は変化しない。

```
while (--n >= 0)  
  文
```

繰返し終了時の*n*の値は-1。

● Fig.4-18 *n*回の繰返しを行う for 文と while 文

上記4つの方法で、同じ*n*回の繰返しを行うプログラムを作ることが可能。ただし、規定回、繰返すループを構成する場合はFOR文を使うほうがよい。

また*i*の初期値を0にするか1にするかについては、一般的には0である。この理由については、次章で説明します。

```
/*
 指示された個数だけ整数を読み込んで合計値と平均値を表示
*/

#include <stdio.h>

int main(void)
{
    int i = 0;
    int sum = 0;          /* 合計値 */
    int num, tmp;

    printf("整数は何個ですか：");
    scanf("%d", &num);

    for (i = 0; i < num; i++) {
        printf("No.%d：", i + 1);
        scanf("%d", &tmp);
        sum += tmp;
    }

    printf("合計値：%d\n", sum);
    printf("平均値：%.2f\n", (double)sum / num);

    return 0;
}
```

実行例

```
整数は何個ですか：6
No.1：65
No.2：23
No.3：47
No.4：9
No.5：153
No.6：777
合計値：1074
平均値：179.00
```

例 iが0のときに1と表示。

多重ループ

List 4-16

chap04/list0416.c

```
/*
  九九の表を表示
*/

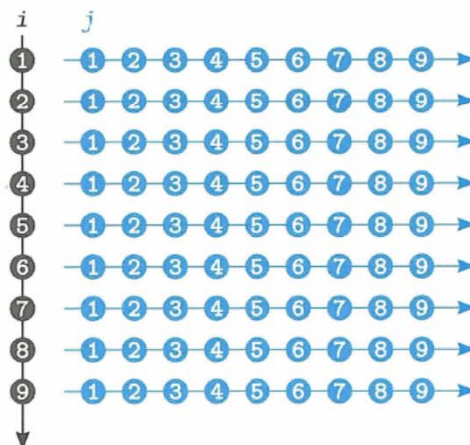
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i <= 9; i++) {
        for (j = 1; j <= 9; j++)
            printf("%3d", i * j);
        putchar('\n');
    }

    return 0;
}
```

変数 i は行に対応。



変数 j は列に対応。

実行結果

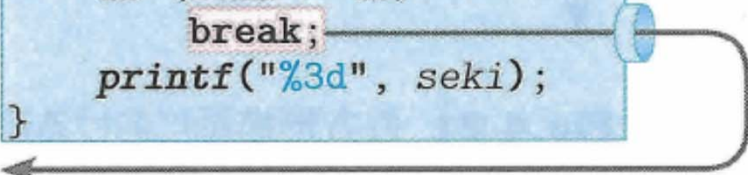
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

● Fig.4-20 九九のプログラムにおける変数の値の変化

■ break 文による繰返しの強制終了

本プログラムの2重ループを以下のように書きかえてみます。そうすると、40以下の値のみが表示されるようになります。

```
for (i = 1; i <= 9; i++) {  
    for (j = 1; j <= 9; j++) {  
        int seki = i * j;  
        if (seki > 40)  
            break;  
        printf("%3d", seki);  
    }  
    putchar('\n');    /* 改行 */  
}
```



chap04/list0416a.c

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	
6	12	18	24	30	36			
7	14	21	28	35				
8	16	24	32	40				
9	18	27	36					

List 4-17

```
/*
  長方形を描画
*/
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i, j;
```

```
    int height, width;
```

```
    puts("長方形を作ります。");
```

```
    printf("高さ : ");    scanf("%d", &height);
```

```
    printf("横幅 : ");    scanf("%d", &width);
```

```
    for (i = 1; i <= height; i++) {
```

```
        for (j = 1; j <= width; j++)
```

```
            putchar('*');
```

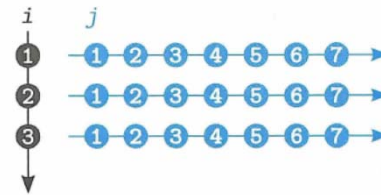
```
        putchar('\n');
```

```
    }
```

```
    return 0;
```

```
}
```

変数iとjの変化



● Fig.4-21 長方形描画における変数の変化

chap04/list0417.c

実行例

長方形を作ります。

高さ : 3

横幅 : 7



```
/* 長方形はheight行 */
```

```
/* 各行にwidth個の'*'を表示 */
```

```
/* 改行 */
```


List 4-18

a 左下が直角の直角二等辺三角形

chap04/list0418.c

```

/*
  左下が直角の直角二等辺三角形
*/
#include <stdio.h>

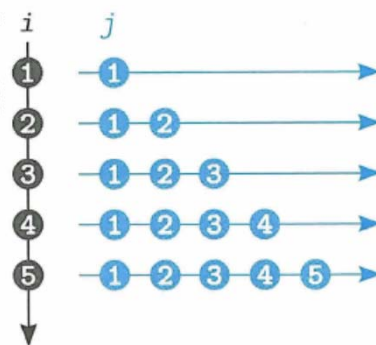
int main(void)
{
    int i, j, len;

    puts("左下直角二等辺三角形を作ります。");
    printf("短辺: ");
    scanf("%d", &len);

    for (i = 1; i <= len; i++) {
        for (j = 1; j <= i; j++)
            putchar('*');
        putchar('\n');
    }

    return 0;
}

```



実行例

左下直角二等辺三角形
を作ります。
短辺: 5



```

/* i行 (i = 1, 2, ..., len) */
/* 各行にi個の '*' を表示 */

/* 改行 */

```

List 4-19

```

/*
 右下が直角の直角二等辺三角
*/
#include <stdio.h>

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

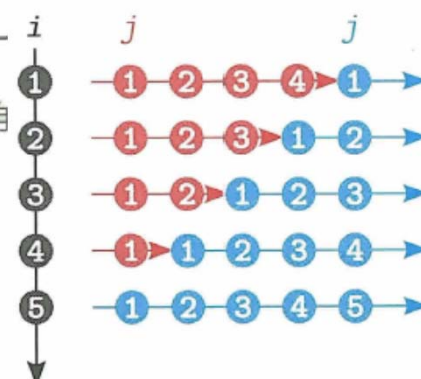
    puts("右下直角二等辺三角形を作ります。");
    printf("短辺: ");
    scanf("%d", &len);

    for (i = 1; i <= len; i++) {
        for (j = 1; j <= len - i; j++)
            putchar(' ');
        for (j = 1; j <= i; j++)
            putchar('*');
        putchar('\n');
    }

    return 0;
}

```

b 右下が直角の直角二等辺三角形



chap04/list0419.c

実行例

右下直角二等辺三角形
を作ります。
短辺: 5

- 赤網部の for 文 … 空白文字 ' ' を表示するための繰返し (表示は $len - i$ 個)。
- 青網部の for 文 … 記号文字 '*' を表示するための繰返し (表示は i 個)。

演習

これができれば、理解度完璧

■ 演習 4-24

右に示すように、読み込んだ整数の段数をもつピラミッドを表示するプログラムを作成せよ。

ヒント：第 i 行目には $(i - 1) * 2 + 1$ 個の '*' 記号を表示することになる。

ピラミッドを作ります。
何段ですか：3

```
*  
***  
*****
```

考え方

- (1) 外側のループは、段数と同じ回数まわす必要あり
- (2) 内側のループは (a)左側の空白を書くループ、(b)真ん中の * を書くループ、(c)右側の空白を書くループに分かれる。合計 段数 \times 2 - 1 回文字 (* か空白を書く必要あり)
- (3) あとは、それぞれのループが何回回ればいいのか？を考える。当然 i に依存した形となる。

(int) (段数-i)

$(i-1)*2+1$

i=1

*

i=2

* * *

i=3

* * * * *

i=4

* * * * *
* * * * *

```
#include<stdio.h>
int main(void)
{
    int i,j,ln;
    printf(" input number of steps:");
    scanf("%d",&ln);

    for(i=1;i<=ln;i++){
        for(j=1;j<=ln-i;j++)    putchar(' ');
        for(j=1;j<=(i-1)*2+1;j++)  putchar('*');
        putchar('\n');
    }
    return(0);
}
```



キーワード

C言語では、`if` や `else` のような語句には、特別な意味が与えられています。このような語句のことを**キーワード** (*keyword*) と呼び、変数名などに利用することはできません。**Table 4-5** に示す 32 個のキーワードがあります。

● **Table 4-5** C言語のキーワード

<code>auto</code>	<code>break</code>	<code>case</code>	<code>char</code>	<code>const</code>	<code>continue</code>
<code>default</code>	<code>do</code>	<code>double</code>	<code>else</code>	<code>enum</code>	<code>extern</code>
<code>float</code>	<code>for</code>	<code>goto</code>	<code>if</code>	<code>int</code>	<code>long</code>
<code>register</code>	<code>return</code>	<code>short</code>	<code>signed</code>	<code>sizeof</code>	<code>static</code>
<code>struct</code>	<code>switch</code>	<code>typedef</code>	<code>union</code>	<code>unsigned</code>	<code>void</code>
<code>volatile</code>	<code>while</code>				

■ インデント

List 4-16 を抜粋した Fig.4-24 をよく見てください。プログラム中の各文は、4 桁ごとに段々が付いています。複合文{ }は、まとまった宣言と文をくくったものであり、いわば日本語での“段落”のようなものです。

段落中の記述を、数桁ずつ右にずらして書くと、プログラムの構造がつかみやすくなります。そのための余白のことを**インデント**（^{だんづ}段付け／^{じき}字下げ）といい、インデントを用いて記述することを**インデントーション**と呼びます。

階層の深さに応じてインデント（段付け／字下げ）する。

```
/*--- 参考：List 4-16より抜粋---*/  
int main(void)  
{  
    → int i, j;  
    → for (i = 1; i <= 9; i++) {  
        → for (j = 1; j <= 9; j++)  
            → printf("%3d", i * j);  
            → putchar('\n');  
        → }  
    → return 0;  
}
```

● Fig.4-24 ソースプログラム中のインデント