第２章
演算と型

もしも「身長と体重を教えてください。」と尋(たず)ねられたら、どのように答えますか。たとえば「身長は 175cm で体重は 60kg です。」と答えたとしましょう。しかし、それは恐らく嘘(？)ですね。身長が 175cm ピッタリということは、まずありません。身長計で計って、175.3cmであったとしても、これも本当の値ではないでしょう。真の値は、おそらく175.2869758…cmといった値のはずです(しかも時間の経過と共に刻一刻と変化しています)。でも、通常は「身長は175cmで体重は 60kg です。」で十分なのです。誰も真に正確な値を知る必要がないのですから。　…　プログラムの世界でも同じです。現実の値を必ずしも正確に表す必要はありません。

この章では、Ｃ言語で数値を扱うために最低限の知識といえる、演算と型の基本について学習します。

2-1 演算

四則演算

前章では、足し算・引き算・掛け算を行いました。今度は、割り算もやってみます。

2

二つの整数値を読み込んで、その和・差・積・商・剰余を表示する。

プログラムを **List 2-1** に示します。

List 2-1

```
/*
    読み込んだ二つの整数値の和・差・積・商・剰余を表示
*/

#include  <stdio.h>

int main(void)
{
    int  vx, vy;

    puts("二つの整数を入力してください。");
    printf("整数vx：");    scanf("%d", &vx);
    printf("整数vy：");    scanf("%d", &vy);

    printf("vx + vy = %d\n",  vx + vy);
    printf("vx - vy = %d\n",  vx - vy);
    printf("vx * vy = %d\n",  vx * vy);
    printf("vx / vy = %d\n",  vx / vy);
    printf("vx %% vy = %d\n", vx % vy);

    return (0);
}
```

実 行 例

```
二つの整数を入力してください。
整数vx：57↵
整数vy：21↵
vx + vy = 78
vx - vy = 36
vx * vy = 1197
vx / vy = 2
vx % vy = 15
```

▶ *vx* - *vy* は、*vx* から *vy* を引いた値を求めるだけであって、必ずしも差を求めるわけではありません。すなわち、*vx* よりも *vy* の方が大きい場合は、*vx* - *vy* は負の値となります。差を求めるプログラムは、**第 3 章**（*p.49*）で示します。

演算子とオペランド

***** が積を求めるための記号であることは、**前章**で解説しました。このような演算の働きをもった記号のことを**演算子**（*operator*）と呼び、その演算の対象となる変数や定数などを**オペランド**（*operand*）と呼びます。たとえば、足し算を行う

```
vx + vy
```

では、演算子は **+** であり、そのオペランドが *vx* と *vy* です。

左側のオペランドを**第 1 オペランド**あるいは**左オペランド**と呼び、右側のオペランドを**第 2 オペランド**あるいは**右オペランド**と呼びます。

▶ C言語には多くの演算子があります。*p.177* には全演算子の一覧表を示しています。

商と剰余

商を求める演算子が **/** です。

整数 / 整数　　商の整数部

という演算では、商の整数部すなわち小数点以下を切り捨てた値が得られます。たとえば 5 / 3 は 1 となり、3 / 5 は 0 となります。一方、

整数 % 整数　　剰　　余

は、**剰余**を求めます。たとえば、5 % 3 は 2 となり、3 % 5 は 3 となります。

▶　次ページ **Column 2-1** も参照してください。

乗除演算子と加減演算子

このプログラムで利用した五つの演算子は、**Table 2-1** に示す**乗除演算子**（*multiplicative operator*）と、**Table 2-2** に示す**加減演算子**（*additive operator*）とに大別されます。

これらの演算子の名称を覚えておきましょう。

Table 2-1　乗除演算子

2項 * 演算子	*a* * *b*	*a*と*b*の積。
/ 演算子	*a* / *b*	*a*を*b*で割った商（整数どうしの場合は小数点以下は切り捨て）。
% 演算子	*a* % *b*	*a*を*b*で割った剰余（*a*と*b*は整数でなければならない）。

Table 2-2　加減演算子

2項 + 演算子	*a* + *b*	*a*と*b*の和。
2項 - 演算子	*a* - *b*	*a*から*b*を引いた値。

▶　それぞれの演算子の英語名は、乗除演算子が *binary * operator*、*/ operator*、*% operator*、加減演算子が *binary + operator*、*binary - operator* です。

printf で % 文字を表示

剰余の表示を行っている箇所に注目しましょう。書式文字列中の **%** は、一つ余分であるように感じられます。しかし、**%d** のように、書式文字列内の文字 % には、書式を指定する変換指定を導くという任務が与えられています。

```
printf("vx %% vy = %d\n", vx % vy);
```

ですから、書式の指定を行うのではなく、本当に % と表示したい場合は **%%** と書くことになっているのです。

▶　書式指定の機能をもたない ***puts*** 関数による表示では、%% としてはいけません（%% と表示されてしまいます）。

最下位の桁を求める

剰余を求める演算子をうまく使うと、次の問題を解くことができます。

> 読み込んだ整数値の最も下の桁の数字を表示する。

プログラムを **List 2-2** に示します。

List 2-2

```
/*
    読み込んだ整数値の最も下の桁の数字を表示
*/

#include  <stdio.h>

int main(void)
{
    int  no;

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

    printf("最も下の桁は%dです。\n", no % 10);

    return (0);
}
```

実行例（1）

```
整数を入力してください：1357
最も下の桁は7です。
```

実行例（2）

```
整数を入力してください：1780
最も下の桁は0です。
```

整数を 10 で割った剰余が、最も下の桁の値となりますね。

● 演習 2-1

右に示すように、二つの整数値を読み込んで、前者の値が後者の何％であるかを表示するプログラムを作成せよ。

```
二つの整数を入力してください。
整数Ａ：54
整数Ｂ：84
Ａの値はＢの64％です。
```

Column 2-1　％演算子とオペランド

剰余を求める％演算子が生成する値は、少し複雑な規則に依ります。

■　二つのオペランドが同符号の値であれば、演算の結果は正の値となります。

正　％　正　→　正
負　％　負　→　正

■　二つのオペランドの符号が異なるときは、演算の結果は処理系に依存することになっていますので注意しましょう。すなわち、処理系が異なれば、得られる値が異なる可能性があります（したがって、このような演算を行うことは、避けた方が無難です）。

正　％　負　→　結果は処理系によって異なる
負　％　正　→　結果は処理系によって異なる

複数の変換指定

二つの整数値を読み込んで、その商と剰余を表示するプログラムが **List 2-3** です。

List 2-3

```
/*
    二つの整数値を読み込んで商と剰余を表示
*/

#include  <stdio.h>

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    printf("ＡをＢで割ると%dあまり%dです。\n", na / nb, na % nb);

    return (0);
}
```

実 行 例

```
二つの整数を入力してください。
整数Ａ：57↵
整数Ｂ：21↵
ＡをＢで割ると2あまり15です。
```

プログラム水色部分の書式文字列には、変換指定%dが二つ含まれています。**Fig.2-1**に示すように、それぞれの変換指定は、左側から第２実引数、第３実引数に対応します。

もちろん、実引数と一対一で対応するのであれば、変換指定は三つ以上あっても構いません。

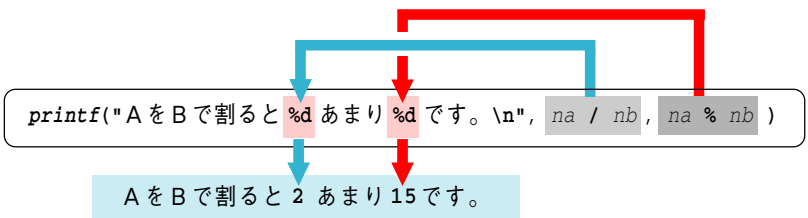

Fig.2-1　printf関数で二つの値を書式化して表示

一度に二つ以上の数値を書式化して表示したいときは、このように書式文字列中に複数の変換指定を書き込みます。

● 演 習 2-2

右に示すように、二つの整数値を読み込んで、その和と積を表示するプログラムを作成せよ。

```
二つの整数を入力してください。
整数Ａ：54↵
整数Ｂ：12↵
それらの和は66で積は648です。
```

単項の算術演算子

今度は、次の問題を考えましょう。

> 読み込んだ整数値の符号を反転した値を表示する。

すなわち、75 と入力されたら -75 と表示し、-64 と入力されたら 64 と表示します。そのプログラムが **List 2-4** です。

List 2-4

```
/*
    読み込んだ整数値の符号を反転した値を表示
*/

#include  <stdio.h>

int main(void)
{
    int  num;

    printf("整数を入力してください：");
    scanf("%d", &num);

    printf("符号を反転した値は%dです。\n", -num);    /* 単項-演算子 */

    return (0);
}
```

実行例（1）

```
整数を入力してください：75
符号を反転した値は-75です。
```

実行例（2）

```
整数を入力してください：-64
符号を反転した値は64です。
```

これまで利用してきた演算子は、二つのオペランドを要するものでした。このような演算子を **2 項**の演算子と呼びます。C 言語の演算子には、オペランドを一つしか必要としない**単項**の演算子や、オペランドを三つ必要とする **3 項**の演算子もあります。

ここで初登場の演算子が、単項の演算子である**単項 - 演算子**（*unary - operator*）です。おそらく説明するまでもないでしょうが、この**-**演算子は、オペランドの値の符号を反転した値を生成します。対（つい）になる演算子として**単項 + 演算子**（*unary + operator*）があり、それらをまとめたものが **Table 2-3** です。

Table 2-3　単項 + 演算子と単項 - 演算子

単項 + 演算子	+a	aの値。
単項 - 演算子	-a	aの符号を反転した値。

▶　すなわち + と - には、2 項版と単項版の 2 種類があるわけです。単項 + 演算子は、実質的には何の演算も行いませんが、単項 - 演算子と対になるように用意されています。

なお、単項 + 演算子、単項 - 演算子、**!** 演算子（*p.61*）、**~** 演算子（*p.164*）の四つの演算子をまとめて**単項算術演算子**（*unary arithmetic operator*）と呼びます。

代入演算子

これまできちんと説明していませんでしたが、いくつかのプログラムでは**Table 2-4**に示す**代入演算子**（*assignment operator*）と呼ばれる = 演算子を使ってきました。

▶ 第4章で解説する**複合代入演算子**との対比から、一般に代入演算子と呼ばれる=は、厳密には**単純代入演算子**（*simple assignment operator*）と呼ばれます。

Table 2-4　単純代入演算子

単純代入演算子　*a* = *b*	*b*を*a*に代入。

式と代入式

変数や定数、さらに、それらを演算子で結合したものを、**式**（*expression*）と呼びます。たとえば、

```
vx + 32
```

では、*vx*, 32, *vx* + 32 のいずれもが式なのです。また、

```
vc = vx + 32                /* 代入式 */
```

では、*vc*, *vx*, 32, *vx* + 32, *vc* = *vx* + 32 のいずれも式とみなすことができます。もちろん、*vc* が代入演算子 = の第１オペランドであり、*vx* + 32 が第２オペランドです。

なお、代入演算子を用いた、このような式のことを**代入式**（*assignment expression*）と呼ぶことを必ず覚えておきましょう。

式文

p.5 で解説したように、原則として、文の末尾には、セミコロン ; が必要ですから、先ほど示した代入式は、以下の形となって、初めて正しい文となります。

```
vc = vx + 32;               /* 式文 */
```

このように、式にセミコロンが付いて文となったものを、**式文**（*expression statement*）と呼びます。

▶ 式文に関しては、**第6章**（*p.137*）でより詳しく解説します。なお、**次章**以降では、式文以外の「文」として、**if** 文や **while** 文などを学習していきます。

2-2 型

平均値を求める

次の問題を考えましょう。

2

二つの整数値を読み込んで、その平均値を求める。

プログラムを **List 2-5** に示します。

List 2-5

```
/*
    二つの整数値を読み込んで平均値を表示
*/

#include  <stdio.h>

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    printf("それらの平均は%dです。\n", (na + nb) / 2);

    return (0);
}
```

実 行 例

```
二つの整数を入力してください。
整数Ａ：40↵
整数Ｂ：45↵
それらの平均は42です。
```

式 *na* + *nb* を囲む ( ) は、演算を優先的に結び付けるための記号です。もし、この式が

```
na  +  nb / 2
```

となっていれば、*na* と *nb* / 2 との和を求めることになります。というのも、私たちが日常利用する計算と同じで、加減算よりも、乗除算の方が優先されるからです。

▶ 全ての演算子の**優先順位**を、*p.177* にまとめています。

型

実行例からもわかりますが、平均値は 42.5 ではなく 42 と表示されます。すなわち、小数点以下の部分は切り捨てられてしまうのです。

整数だけしか扱えないというのは、**int** という**型**（*type*）の性質なのです。

int 型と double 型

C 言語では、実数を**浮動小数点数**（*floating-point number*）という形式で表します。その型にはいくつかの種類がありますが、ここでは**double**型を学習します。整数である**int**型と浮動小数点数である **double** 型との違いを **List 2-6** のプログラムで確認しましょう。

List 2-6

```
/*
    整数と浮動小数点数
*/

#include  <stdio.h>

int main(void)
{
    int      nx;      /* 整数 */
    double   dx;      /* 浮動小数点数 */

    nx = 9.99;
    dx = 9.99;

    printf(" int   型変数nxの値：%d\n", nx);          /*    9       */
    printf("              nx / 2：%d\n", nx / 2);     /*    9 / 2   */

    printf("double型変数dxの値：%f\n", dx);          /* 9.99       */
    printf("              dx/2.0：%f\n", dx / 2.0);   /* 9.99 / 2.0 */

    return (0);
}
```

実行結果

```
 int   型変数nxの値：9
              nx / 2：4
double型変数dxの値：9.990000
              dx/2.0：4.995000
```

変数 *nx* を **int** 型、変数 *dx* を **double** 型として宣言して、それらに 9.99 を代入しています。**Fig.2-2**に示すように、**int**型変数に実数値を代入するときは、その小数点以下の部分は切り捨てられるのです。したがって、*nx* に格納される値は 9 となります。

もちろん、*nx* **/** 2 すなわち 9 **/** 2 は、整数 / 整数という演算ですから、その結果も小数点以下の部分が切り捨てられたものとなります（*p.19*）。

なお、***printf*** 関数で **double** 型の値を表示するための変換指定は、**"%d"** ではなく **"%f"** となることに注意しましょう。

▶　変換指定 **"%f"** の f は、浮動小数点 *floating-point* の頭文字です。小数点以下の部分は 6 桁も表示されますが、この桁数を変更する方法は *p.32* で解説します。

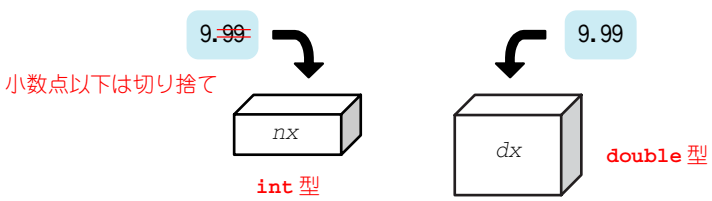

Fig.2-2　整数と浮動小数点数

型とオブジェクト

型と変数について、もう少し詳しく学習しましょう。

Fig.2-3 では、型である **int** 型と **double** 型を点線の箱で表しており、それらの型をもつ変数 *nx* と変数 *dx* を実線の箱で表しています。型を表す点線の箱は、その変数と同じ大きさとなっていますね。

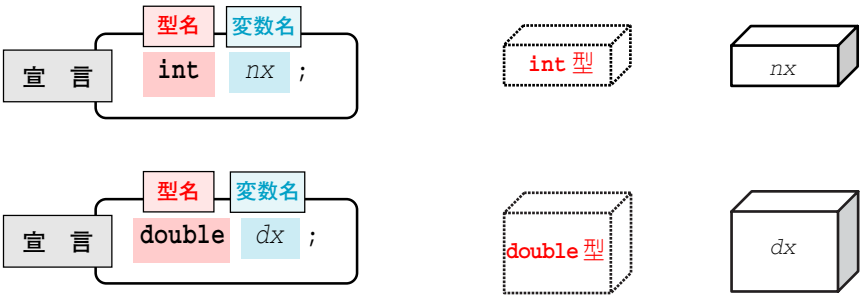

Fig.2-3 型とオブジェクト

前ページのプログラムからもわかりますが、**int** 型は、整数のみを表現することができます。実数値が代入されようとしても、その整数部分だけを受け入れます。

一方、浮動小数点型の一つである **double** 型は、小数点以下の部分をもつ実数値を表現することができます。

第 7 章で詳しく解説しますが、C 言語には、多くの型が用意されています。

ただし、各型は、値を無限に表現できるわけではありません。たとえば、**int** 型が表現できる値として保証されているのは、-32,767 から 32,767 までです。

▶ 処理系によっては、より広い範囲の数を表現できます。*p.153* を参照してください。

それぞれの型には固有の性質がありますが、その性質をそっくり受け継いで作られた実体である変数のことを、正式には**オブジェクト** (*object*) と呼びます。すなわち、次のように理解しましょう。

■ 重 要 ■

型とは、その諸性質を内に秘めた設計図（タコ焼きのカタ）であり、その型をもつオブジェクト（変数）は、その設計図をもとに作られた実体（カタから作られた本物のタコ焼き）である。

なお、変数という言葉の方が一般的であり、柔らかい感じですので、本書では、厳密さが要求されないところでは、オブジェクトではなく変数という言葉を使っていくことにします。

整数定数と浮動小数点定数

プログラムに直接埋め込まれる値である定数にも型があります。5 や 37 などの定数は、整数の型をもちますから**整数定数**（*integer constant*）と呼ばれ、3.14 のような小数点以下の部分をもつ定数は、**浮動小数点定数**（*floating constant*）と呼ばれます。

基本的には、整数定数は **int** 型であり、浮動小数点定数は **double** 型です。

▶　値の大きさや、特別な指示によって、型が変わります。**第 7 章**を参照してください。

double 型の演算

二つの実数値を読み込んで、それらの和・差・積・商の四つの値を表示するプログラムを **List 2-7** に示します。

List 2-7

```
/*
    二つの実数値を読み込んで和・差・積・商を実数で表示
*/

#include  <stdio.h>

int main(void)
{
    double  vx, vy;      /* 浮動小数点数 */

    puts("二つの数を入力してください。");
    printf("実数vx：");    scanf("%lf", &vx);
    printf("実数vy：");    scanf("%lf", &vy);

    printf("vx + vy = %f\n", vx + vy);
    printf("vx - vy = %f\n", vx - vy);
    printf("vx * vy = %f\n", vx * vy);
    printf("vx / vy = %f\n", vx / vy);

    return (0);
}
```

実 行 例

```
二つの数を入力してください。
実数vx：40.5↵
実数vy：5.2↵
vx + vy = 45.700000
vx - vy = 35.300000
vx * vy = 210.600000
vx / vy = 7.788462
```

Table 2-5 にまとめていますが、**double** 型の変数に値を読み込むときに ***scanf*** 関数に与える変換指定は **"%lf"**（エル）となります。要注意です。

Table 2-5　変換指定

	int型	double型
printf	*printf*("%d", *no*)	*printf*("%f", *no*)
scanf	*scanf*("%d", &*no*)	*scanf*("%lf", &*no*)

● 演 習 2-3

右に示すように、読み込んだ実数値をそのまま表示するプログラムを作成せよ。

```
実数を入力してください：57.3↵
あなたは57.300000と入力しましたね。
```

型と演算

整数 / 整数という演算によって得られるのは、商の小数点以下を切り捨てた値ですが、浮動小数点数どうしの演算では、そのような切り捨ては行われません。

▶ なお % 演算子は、その性格のため、整数どうしの演算にのみ使うことができ、浮動小数点数の演算には利用できません。

2

Fig.2-4に示していますが、`int / int` や `double / double` のように、両方のオペランドの型が同じである演算では、それによって得られる値の型は、演算の対象となるオペランドと同じ型になります。

また、`int / double` や `double / int` のように、一方のオペランドが**int**型で、他方のオペランドが**double**型である場合、**int**型のオペランドの値が**double**型に"格上げ"されるという**暗黙の型変換**が行われ、**double**型どうしの演算として行われます。したがって、演算によって得られる結果も**double**型となります。

もちろん、ここに示した規則は、+ や * など他の演算にも適用されます。

```
  5    /  2      →  2          5.0   / 2.0       →  2.5
 int   / int    →  int       double / double    → double
```

同じ型どうしの演算

```
  5.0   /  2                    5    / 2.0
double  / int                  int    / double
          ↓ 格上げ               ↓ 格上げ
  5.0   /  2.0    →  2.5       5.0   / 2.0       →  2.5
double  / double  → double    double / double    → double
```

異なる型の演算

Fig.2-4　演算と型

C言語には、たくさんの型がありますので、細かい規則は複雑なのですが、大まかには以下のように理解しておきましょう（詳しい規則は*p.179* に示しています）。

■ 重 要 ■

演算の対象となるオペランドの型が異なるとき、小さい方の型のオペランドは、より大きい（懐（ふところ）の広い）方の型に変換された上で演算が行われる。

▶ ここで、"大きい"という表現を使っていますが、必ずしも**double**型が、物理的に**int**型より大きいというわけではありません。小数点以下の部分を格納する余裕があるという意味です。

List 2-8 に示すプログラムで、前ページに示した規則を確認しましょう。

List 2-8

```
/*
    型と演算について確認する
*/

#include  <stdio.h>

int main(void)
{
    int     n1, n2, n3, n4;     /* 整数 */
    double  d1, d2, d3, d4;     /* 浮動小数点数 */

    n1 = 5   / 2;           /* n1 ← 2   */
    n2 = 5.0 / 2.0;         /* n2 ← 2.5   （代入時に小数点以下切り捨て） */
    n3 = 5.0 / 2;           /* n3 ← 2.5   （代入時に小数点以下切り捨て） */
    n4 = 5   / 2.0;         /* n4 ← 2.5   （代入時に小数点以下切り捨て） */

    d1 = 5   / 2;           /* d1 ← 2    */
    d2 = 5.0 / 2.0;         /* d2 ← 2.5  */
    d3 = 5.0 / 2;           /* d3 ← 2.5  */
    d4 = 5   / 2.0;         /* d4 ← 2.5  */

    printf("n1 = %d\n", n1);
    printf("n2 = %d\n", n2);
    printf("n3 = %d\n", n3);
    printf("n4 = %d\n\n", n4);

    printf("d1 = %f\n", d1);
    printf("d2 = %f\n", d2);
    printf("d3 = %f\n", d3);
    printf("d4 = %f\n", d4);

    return (0);
}
```

実行結果

```
n1 = 2
n2 = 2
n3 = 2
n4 = 2

d1 = 2.000000
d2 = 2.500000
d3 = 2.500000
d4 = 2.500000
```

int 型である変数 *n1* には 2 が代入され、*n2*, *n3*, *n4* には 2.5 が代入されることになります。もっとも、代入時には小数点以下の部分は切り捨てられますので、これら四つの変数には、結局 2 が格納されることになります。

また、**double** 型である変数 *d1* には 2 が、*d2*, *d3*, *d4* には 2.5 が代入され、それらの値が格納されます。

▶ 浮動小数点数は、無限の精度をもつわけではありませんから、全ての桁を正確に表現できるとは限りません。*p.172* を参照してください。

● 演習 2-4

整数定数、浮動小数点定数、**int** 型の変数、**double** 型の変数を、掛けたり割ったりして、いろいろな演算を行うプログラムを作成し、本文に示した規則を確認せよ。

キャスト

List 2-5 に示した二つの整数値の平均値を求めるプログラムは、平均値の整数部分のみを表示するものです。今度は、小数点以下の部分も表示するようにしましょう。そのプログラムを **List 2-9** に示します。

List 2-9

```
/*
    二つの整数値を読み込んで平均値を実数で表示
*/

#include  <stdio.h>

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    printf("それらの平均は%fです。\n", (na + nb) / 2.0);

    return (0);
}
```

実 行 例

```
二つの整数を入力してください。
整数Ａ：40↵
整数Ｂ：45↵
それらの平均は42.500000です。
```

平均を求める式に着目しましょう。

```
( na + nb ) / 2.0
```

最初に、( )で囲まれた水色部分の演算が行われます。この演算は、**int + int** 、すなわち、**整数 + 整数** ですから、その演算結果も整数である **int** 型となります。

したがって、

```
( 整数 ) / 2.0
```

となるわけです。これは、

```
整数  /  実数
```

ですから、その結果も実数となります。

このようにして、平均値を実数値として求めているのです。

しかし、日常生活で平均を求めるときに、私たちは「2で割ろう。」と考えるのであって、「2.0 で割ろう。」とはあまり思わないはずです。

二つの整数の和の方を、いったん実数に変換し、それを2で割ることによって平均を求めるプログラムが **List 2-10** です。

List 2-10

```
/*
    二つの整数値を読み込んで平均値を実数で表示（キャスト）
*/

#include  <stdio.h>

int main(void)
{
    int  na, nb;

    puts("二つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);

    printf("それらの平均は%fです。\n", (double)(na + nb) / 2);  /* キャスト */

    return (0);
}
```

実行例

```
二つの整数を入力してください。
整数Ａ：40↵
整数Ｂ：45↵
それらの平均は42.500000です。
```

一般に、

```
( 型 ) 式
```

という形の式は、式の値を型としての値に変換したものを生成します。

このような**明示的な型変換**を**キャスト**（*cast*）と呼び、( )を**キャスト演算子**（*cast operator*）と呼びます（**Table 2-6**）。

Table 2-6　キャスト演算子

キャスト演算子	（型名）*a*	*a*の値を型名で指定された型の値に変換したものを生成。

したがって、平均を求める過程では、まず最初に、

```
(double)(na + nb)
```

によって、*na* + *nb* の値を**double**型で表現した値に変換したものを生成します（たとえば整数85から浮動小数点数85.0が作られます）。式(*na* + *nb*)の値は、**double**型へとキャストされますので、平均を求める式は、

```
実数 / 整数
```

となります。こうやって、平均値を実数として求めているのです。

● 演習 2-5

右に示すように、二つの整数値を読み込んで、前者の値が後者の何％であるかを**実数**で表示するプログラムを作成せよ。

```
二つの整数を入力してください。
整数Ａ：54↵
整数Ｂ：84↵
Ａの値はＢの64.285714％です。
```

変換指定

三つの整数値を読み込んで、その合計値と平均値を表示するプログラムを**List 2-11**に示します。

List 2-11

```
/*
    三つの整数値を読み込んで合計値と平均値を表示
*/

#include  <stdio.h>

int main(void)
{
    int     na, nb, nc;
    int     sum;            /* 合計値 */
    double  ave;            /* 平均値 */

    puts("三つの整数を入力してください。");
    printf("整数Ａ：");    scanf("%d", &na);
    printf("整数Ｂ：");    scanf("%d", &nb);
    printf("整数Ｃ：");    scanf("%d", &nc);

    sum = na + nb + nc;
    ave = (double)sum / 3;      /* キャスト */

    printf("それらの合計は%5dです。\n",    sum);     /* 99999 */
    printf("それらの平均は%5.1fです。\n",  ave);     /* 999.9 */

    return (0);
}
```

実 行 例

```
三つの整数を入力してください。
整数Ａ：87↵
整数Ｂ：45↵
整数Ｃ：59↵
それらの合計は   191です。
それらの平均は  63.7です。
```

このプログラムを理解するポイントは、***printf*** 関数に渡している書式文字列中の二つの変換指定 **%5d** と **%5.1f** です。これらは、以下のような意味です。

%5d … 整数を 10 進数で少なくとも 5 桁で表示。

%5.1f … 浮動小数点数を少なくとも 5 桁で表示。ただし、小数点以下は 1 桁。

もう少し詳しく説明しましょう。変換指定は、以下のような形式になっています。

%09.9f

次ページの **List 2-12** と、その実行結果を対比しながら理解してください。

(a) 0 フラグ

0 が指定されていると、数値の前に余白があるときは、0 が詰めて表示されます（省略した場合は空白が詰められます）。

(b) 最小フィールド幅

少なくとも、この桁数だけの表示が行われます。この指定が省略された場合および実際に表示する数値が指定された値を超えるときは、その数値を表示するのに必要な桁数で表示されます。なお、-が指定された場合は左側に詰めて表示され、指定がない場合は右側に詰められます。

List 2-12

```
/*
    整数と浮動小数点数を書式化して表示
*/

#include  <stdio.h>

int main(void)
{
    printf("[%d]\n",          123);
    printf("[%.4d]\n",        123);
    printf("[%4d]\n",         123);
    printf("[%04d]\n",        123);
    printf("[%-4d]\n\n",      123);

    printf("[%d]\n",          12345);
    printf("[%.3d]\n",        12345);
    printf("[%3d]\n",         12345);
    printf("[%03d]\n",        12345);
    printf("[%-3d]\n\n",      12345);

    printf("[%f]\n",          123.13);
    printf("[%.1f]\n",        123.13);
    printf("[%6.1f]\n\n",     123.13);

    printf("[%f]\n",          123.13);
    printf("[%.1f]\n",        123.13);
    printf("[%4.1f]\n\n",     123.13);

    return (0);
}
```

実行結果

```
[123]
[0123]
[ 123]
[0123]
[123 ]

[12345]
[12345]
[12345]
[12345]
[12345]

[123.130000]
[123.1]
[ 123.1]

[123.130000]
[123.1]
[123.1]
```

(c) 精度

表示する最小の桁数を指定します。省略した場合は、整数の場合は1、浮動小数点数の場合は 6 であるとみなされます。

(d) 変換指定子

d … **int** 型の整数を 10 進数で表示します。

f … **double** 型の浮動小数点数を 10 進数で表示します。

▶ ここに示した変換指定の仕様は、ごく一部です。***printf***関数に関する詳細は、*p.318*を参照してください。

● 演習 2-6

右に示すように、身長を整数値として読み込んで、標準体重を実数で表示するプログラムを作成せよ。標準体重は（身長 - 100）× 0.9 によって求め、その小数点以下は、1 桁だけ表示すること。

```
身長を入力してください：175
標準体重は67.5です。
```