4　式と演算子

定数 (constant)、オブジェクト (object, 変数とも呼ばれる)、及びこれらと演算子 (operator) を並べたものが式 (expression) である。これらを順に見ていく。

4.1　定数

定数には浮動小数点定数、整数定数、文字定数など[9] がある。定数は型を持つ。プログラム実行中に値は変わらない。

4.1.1　整数定数

整数定数 (integer constant) には、10 進定数、8 進定数、16 進定数がある。ピリオドおよび指数部を持たない。型を示す接尾語 u, U, l, L, ll, LL が付くことがある。

種類	表し方	例
10 進定数	0(ゼロ) 以外の数字から始まる 10 進数字列	97
8 進定数	接頭語 0(ゼロ) の 8 進数字列	0141
16 進定数	接頭語 0x(ゼロ, エックス) または 0X の 16 進数字列	0x61

例 4.1 演習 2.4(p.16) の

```
1   /*****************************************
2   * add1.c                                  *
3   *****************************************/
4   #include <stdio.h>
5
6   int main(void)
7   {
8     int a,b,sum;
9
10    a=3;
11    b=4;
12    sum=a+b;
13    printf("a+b=%d\n",sum);
14
15    return 0;
16  }
```

の 10 行目の 3、11 行目の 4、15 行目の 0 は整数定数である。

例 4.2 例 3.5(p.26) の 123456789012LL は long long int 型整数定数である。

4.1.2　文字定数

文字定数 (character constant) は文字または逆斜線表記を持つ文字を一重引用符' で囲って表す。int 型で、値は char 型のオブジェクトを int 型に変換したときの値である。

演習 4.1 3.2.2 節 (p.23) で取り上げた char.c の 8-9 行目を `c='a';` で置き換えたプログラム char1.c をつくり実行せよ。`'a'` は文字定数である。

[9] 列挙定数があるが後期のコンピュータ IIIA で扱う。

4.1.2.1　逆斜線表記　キーボードで入力できない制御文字や特別の目的で利用される記号などは 3.2.1 節 (p.22) で述べた逆斜線表記 (エスケープシークエンス escape sequence) を用いる。逆斜線表記には p.22 の表以外にも下記のものがある。

逆斜線表記	意味	内部表現 (16 進)
\”	二重引用符	22
\’	一重引用符	27
\?	疑問符	3F
\\	バックスラッシュ	5C
$\backslash o_1$, $\backslash o_1 o_2$, $\backslash o_1 o_2 o_3$ (o_1 などは 8 進数字)	8 進拡張表記 (\0∼\377)	00∼FF
$\backslash x h_1$, $\backslash x h_1 h_2$ (h_1 などは 16 進数字)	16 進拡張表記 (\x0∼\xff)	00∼FF

逆斜線表記は文字定数や文字列リテラルの中で用いることができる。

例 4.3　1. `'\n'` は改行の文字定数。

2. `'\''` は一重引用符の文字定数。

3. `"\""` は二重引用符の文字列リテラル。

演習 4.2 p.22 で述べた逆斜線表記 (\a∼\r) を一重引用符’で囲った文字定数を%x と%c で表示するプログラムをつくれ。(制御の動作が分かるように変換指定子の後に適当な 1 文字 (例えば、*) を置け。)

演習 4.3 次の 4 つの文は同じ動作 (改行) をすることを確かめよ。

1. `printf("\n");`
2. `printf("%c",'\n');`
3. `printf("%c",'\012');`
4. `printf("%c",'\xa');`

4.1.2.2　8 進整数定数と 8 進拡張表記　8 進整定数は 0 から始まり 0 から 7 までの数字の列 (何個でも良い) である。たとえば、03732 は 10 進整数定数の $2010 (= 3 \times 8^3 + 7 \times 8^2 + 3 \times 8 + 2)$ と同じである。

8 進拡張表記は文字定数を表すためのもので、char 型の 8 ビットを 8 進数で表し先頭に逆斜線 \ をつける。たとえば、改行文字`'\n'` は

0	0	0	0	1	0	1	0

であるので、8 進拡張表記は`'\012'` または`'\12'` である。一方、'A' は

0	1	0	0	0	0	0	1

であるので、8 進拡張表記は`'\101'` である。これを`'\0101'` と表すことはできない。

4.1.3　浮動小数点定数

浮動小数点定数[10]は、有効数字部 (significant part) を持つ。指数部 (exponent part) と型を指定する接尾語がついてもよい。有効数字部は整数部または小数部のいずれかがなければならず、ピリオドまたは指数部のいずれかがなければならない。

たとえば、3.14、6.02e+23 は浮動小数点定数で、3.14、6.02 は有効数字部、e+23 は指数部である。

浮動小数点定数は、1.0f(1.0F) のように接尾語 f(F) を付けると float 型定数、1.0ℓ(1.0L) のように接尾語 ℓ(L) を付けると long double 型定数、接尾語がないときは double 型定数として扱われる。

[10] ANSI C では 10 進浮動小数点定数のみであったが、C99 では 16 浮動小数点定数も加わった。16 浮動小数点定数の有効数字部は 16 進有理数で、指数部は 2 のべき乗である。たとえば、3.141592= $1.10010010000111111011_{(2)} \times 2^1$ は 0x1.921fbp+1 となる。(a 変換)

型	例
float	1.0f、3.14f、3.14F、6.02e+23f、 6.02E+23F
double	1.0、3.141592653589793、3.14、6.02e+23
long double	1.0ℓ、1.0L、3.141592653589793L、3.14L、6.02E+23L

注 4.1 -3.14 は浮動小数点定数 3.14 にマイナス演算子が作用した式と考える。

4.2 オブジェクト

オブジェクトは 2.7 節で説明をしたように型と識別子を持つ。オブジェクトを用いるときは宣言を行う。オブジェクトの宣言は、

```
型 識別子;
```

とする。同じ型のオブジェクトが複数あるときは、カンマ',' で区切って並べることができる。たとえば同じ型の 3 つのオブジェクトを宣言するときは以下のようにする。

```
型 識別子, 識別子, 識別子;
```

たとえば、add1.c の 8 行目は、int 型オブジェクト a、b、sum の宣言である。

4.3 演算子とオペランド

演算子について、これまでに加算演算子 (p.14)、代入演算子 (p.14)、sizeof 演算子 (p.26) にふれた。

定数やオブジェクトなどの式に演算を行うものが演算子 (operator) である。演算子が作用するものをオペランド (operand) という。オペランドに演算子が作用したものは式である。

例 4.4 add.c の a+b において + は加算演算子、a と b はオペランドである。a+b は式である。sum=a+b の = は代入演算子、sum と a+b はオペランドである。sum=a+b は式である。

演算子にはオペランドを 1 つとる単項演算子、2 つとる 2 項演算子、とそれ以外がある。sizeof 演算子は単項演算子、加算演算子と代入演算子は 2 項演算子である。

4.4 単純代入演算子

単純代入演算子 (simple assingnment operator) の形式は次の通り。

```
オブジェクトの識別子=式;
```

右オペランドの式の値を左オペランドの型に変換し、左オペランドの識別子が指し示すオブジェクトに代入する。定数、オブジェクトは式である。

```
1  /* assignment.c */
2  #include <stdio.h>
3
4  int main(void)
5  {
6    int n;
7
8    n=1024;
9    printf("n の値は%d です。\n",n);
10   n=3.14;
11   printf("n の値は%d です。\n",n);
12
13   return 0;
14 }
```

6 行目で int 型オブジェクト n を宣言し、8 行目で n に整数定数 1024 を代入し、10 行目で n に 3.14 を代入している。浮動小数点定数 3.14 は左オペランドの型 int 型に変換され n の値は 1024 が 3 に上書きされる。実行結果は、

```
% cc -Wall assignment.c -o assignment
% ./assignment
n の値は 1024 です。
n の値は 3 です。
```

オブジェクトに値を代入するには、単純代入演算子=以外に複合代入演算子がある。これについて 4.7.1 節で扱う。

4.4.1 初期化

オブジェクトの宣言の際格納する値を指定することを初期化 (initialization) という。

型 オブジェクトの識別子=式;

単純代入演算子の右オペランドを初期化子という。assignment.c の 6-8 行目は初期化を用いると

```
int n=1024;
```

と 1 つの文で書ける。

4.5 算術演算子

add.c の a+b において + は加算演算子、subtraction.c の- は減算演算子である。同様に乗算演算子*、除算演算子/などが与えられており、これらの演算子を算術演算子という。

演算子+、 -、 *は、オペランドの型による違いはないが、除算演算子/はオペランドの型により結果が異なる。

4.5.1 オペランドが整数型

整数型オペランド (定数とオブジェクト) に対する算術演算子 (arithmetic operator) には次の 6 つがある。('==' は、左辺と右辺が等しいことを表わす関係演算子である。)

演算子	優先度	名前	意味	例
-	1	単項のマイナス演算子	符号を変える	-(-3)==3
*	2	乗算演算子	2 数の積	3*5==15
/	2	除算演算子	商の整数部分 †	3/5==0, 5/3==1, (-5)/3==-1, 5/(-3)==-1
%	2	剰余演算子	除算の余り	3%5==3, 5%3==2, (-5)%3==-2 , 5%(-3)==2
+	3	加算演算子	2 数の和	3+5==8
-	3	減算演算子	2 数の差	3-5==-2

† 除算演算子のオペランドの一方が負の値の結果は、処理系により異なる。((-5)/3==-1 となる場合と、(-5)/3==-2 となる場合がある。) a/b が表現できる場合、剰余演算子%は、(a/b)*b+a%b==a を満たすように定義される。上記の表は東京女子大のシステムの例である。

演習 4.4 除算演算子/と剰余演算子%の関係は、2 つのオペランド a と b が正の int 型のとき、

```
(a/b)*b+a%b==a
```

となることを確かめよ。(Hint. $a = bq + r, r \geqq 0$ とおき左辺を計算してみよ。)

優先度が同じ場合は左から右へ行う。括弧'(' と')' により、優先度 (評価順位) を変更できる。

```
1+2*3==7,  (1+2)*3==9
```

演習 4.5 add.c を+, -, *, /, % の演算結果が出力できるように修正したプログラム arithmetic.c を作成せよ。(printf 文で % を出力するには、%% とする。) arithmetic.c を用いて、上の表に示した例を確認せよ。

4.5.2 オペランドが浮動小数点

浮動小数点型オペランド (定数とオブジェクト) に対する算術演算子 (arithmetic operator) には次の 5 つがある。

演算子	優先度	名前	意味	例
-	1	単項のマイナス演算子	符号を変える	-(-3.4)==3.4
*	2	乗算演算子	2 数の積	3.4*5.3==18.02
/	2	除算演算子	2 数の商	3.4/5.3==0.641509
+	3	加算演算子	2 数の和	3.4+5.3==8.7
-	3	減算演算子	2 数の差	3.4-5.3==-1.9

演習 4.6 double 型データに対し、+, -, *, /の演算結果が出力できるように演習 4.5 を修正したプログラム darithmetic.c を作成せよ。darithmetic.c を用いて、上の表に示した例を確認せよ。

4.5.3 オペランドの型が異なる

2 項演算子の 2 つのオペランドの型が異なる場合は共通の型に変換[11]したあと、演算を行う。一方が型 double を持つときは、他方のオペランドを double に変換する。両方とも double でなく、一方が型 float を持つときは、他方のオペランドを float に変換する。

double 型 > float 型 > int 型

の大きい方の型に合わせる。

例 4.5 (1) 5/3==1, 5.0/3==1.666667, 5/3.0==1.666667
(2) 5.0%3 と 5%3.0 は定義できない。

演習 4.7 次の式の型と値を求めよ。
(1) 9+4 (2) 9.0+4 (3) 9+4.0f (4) 3.14f*2 (5) 9/4 (6) 9.0/4 (7) 9%4 (8) 3*4+5
(9) 3*(4.0+5) (10) 0.002*100

[11] 通常の算術型変換 (usual arithmetic conversion) という。

演習 4.8 商品代を入力し消費税を出力するプログラム tax.c を作成せよ。ただし、消費税は 5%で 1 円未満は切り捨てるものとする。(Hint. 5%は 1/20 である。)

演習 4.9 底辺の長さと高さを入力し、三角形の面積を求めるプログラム triangle.c を作成せよ。(辺の長さが小数値であっても計算できるようにすること)

演習 4.10 温度の単位に摂氏 (Celsius) と華氏 (Fahrenheit) がある。摂氏 C°C と華氏 F°F の間には、

$$C = \frac{5}{9}(F - 32), \quad F = \frac{9}{5}C + 32$$

の関係がある。華氏を入力し、摂氏を出力するプログラム fahr2cel.c を作成せよ。出力例として 911°F は何 °C になるかを求めよ。

4.6 論理演算子

4.6.1 論理型

ANSI C では論理型は定義されていないので、int 型で真を 1、偽を 0 とする。C99 では`_Bool` 型[12] が定義されている。

整数型や浮動小数点型の定数やオブジェクトは値が 0 以外のとき真、値が 0 のとき偽とする。

例 4.6 定数 (int 型、double 型、文字型) の真偽の例を示す。

- 1、-1、123、3.14、'a' は真である。
- 0、0.0、'\0' は偽である。

4.6.2 論理演算子

論理演算子を次表に示す。論理演算子のオペランドは整数型や浮動小数点型など[13] である。

論理演算子	名称	意味
`&&`	論理 AND 演算子	AND(論理積)
`||`	論理 OR 演算子	OR(論理和)
!	論理否定演算子	NOT(否定)

論理演算子の優先順位は高い方から、!、&&、||となる。括弧'(' と')' で囲まれてる式は先に評価する。

論理演算子は次の真理表 (truth table) に従う。p と q は真 (true) のときは 1、偽 (false) のときは 0 とする。

p	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

[12] 整数型や浮動小数点型を_Bool 型に変換する場合、その値が 0 に等しいときは結果は 0、0 に等しくないときは結果は 1 とする。
[13] 厳密にはスカラ型である。

注 4.2 演算子とオペランドの間に空白をつけてもよいが、&& の間に空白をつけてはならない。|| についても同様。

演習 4.11 論理 AND 演算子は、第 1 オペランドの値が 0 に等しい場合、第 2 オペランドは評価しない。論理 OR 演算子は、第 1 オペランドの値が 0 に等しくない場合、第 2 オペランドは評価しない。理由を考えよ。

例 4.7 真理表をプログラムを作り確かめる。

```
/* truth-table.c */
#include <stdio.h>

int main(void)
{
  int T=1,F=0; /* T を真、F を偽に初期化 */

  printf("F&&F=%d F||F=%d !F=%d \n",F&&F, F||F,!F);
  printf("F&&T=%d F||T=%d !F=%d \n",F&&T, F||T,!F);
  printf("T&&F=%d T||F=%d !T=%d \n",T&&F, T||F,!T);
  printf("T&&T=%d T||T=%d !T=%d \n",T&&T, T||T,!T);
}
```

```
% cc -Wall truth-table.c -o truth-table
% ./truth-table
F&&F=0 F||F=0 !F=1
F&&T=0 F||T=1 !F=1
T&&F=0 T||F=1 !T=0
T&&T=1 T||T=1 !T=0
```

演習 4.12 例 4.6 をプログラムを作り確認せよ。
Hint. 3.14 の真偽は `int T=1;` のとき `T&&3.14` の値が 1 か 0 かで判断できる。

演習 4.13 次の式の値を求めよ。
(1) `1||0` (2) `!1&&0` (3) `!(1&&0)` (4) `0&&(1||0)` (5) `0||(1&& 0)` (6) `(0||1)&&(1||0)`

演習 4.14 次の 2 式は等しいことを真理表を作り確かめよ。
(1) `!(p&&q)` と `(!p||!q)`
(2) `!(p||q)` と `(!p&&!q)` (de Morgan)
(3) `p&&!q||q&&!p` と `(p||q)&&!(p&&q)`

注 4.3 演習 4.14(3) の式は p と q の排他的論理和 (exclusive or, XOR) という。

論理演算子のオペランドが算術式である場合、まず算術式の算術演算を行い、算術演算の結果が非ゼロならば 1、ゼロならば 0 として論理演算を行う。

例 4.8 論理 AND 式 3+1 && 2*3 は、4 && 6 と等価で、4 も 6 はゼロでないので、3+1 && 2*3 の値は 1 である。

演習 4.15 次の式の値を求めよ。
(1) `!(2+3)` (2) `2+3 && 2-3+1` (3) `2+3 || 2-3+1` (4) `!!(1+2)`

注 4.4 `p` が 0 または 1 と場合は、`!!p` は `p` と等しいが、式の場合は必ずしも等しくない。

4.6.3 関係演算子と等価演算子

関係演算子 (relational operator) と等価演算子 (equality operator) のオペランドは論理演算子と同様整数型や浮動小数点型[14]である。関係演算子と等価演算子は、指定された関係が真の場合は 1 を、偽の場合は 0 を返す。

関係演算子	意味	真の例	等価演算子	意味	真の例
>	より大きい	`10 > 9`	==	等しい	`10 == 10`
<	より小さい	`9 < 10`	!=	等しくない	`9 != 10`
>=	以上	`10 >= 10`			
<=	以下	`9 <= 10`			

(これまでに取り上げた) 演算子の優先順位は次の表の通り。

優先順位	演算子
高い	算術演算子
	!
↕	> >= < <=
	== !=
	&&
低い	||

たとえば、`(10 > 9) && (1 == 1)` を考える。`> && ==`の中で`&&`が最も優先順位が低いので、`10 > 9 && 1 == 1` と () を省略することができる。`1 && 1` と同じだから値は 1 である。

演習 4.16 次の式の値を求めよ。(演算子の優先順位に気をつけること)

(1) `2.1 > 3.2` (2) `2.1 != 2` (3) `2.1 <= 3.2` (4) `2.1 > 1.2 && 3.2 == 3.2`

(5) `2.1 > 1.2 || 2.1 < 1.2` (6) `3 > 0.8 + 1.5 && 3.2 == 4 * 0.8` (7) `-2 < -1 < 0`

(8) `3 < 2 < 4` (9) `1 < 2 == 3 < 4` (10) `!(1.2 < 2.3)`

注 4.5 演習 4.16 の (7)(8)(9) のように数学の式と一致しないことがあるので注意を要する。
C の式 `-2 < -1 < 0` は、`(-2 < -1) < 0` すなわち `1 < 0` を評価することになるが、数学の不等式 $-2 < -1 < 0$ は 「$-2 < -1$ かつ $-1 < 0$」の意味である。

例 4.9 関係演算子と等価演算子の値を表すプログラム。

```
/* test-bool.c */
#include <stdio.h>

int main(void)
{
  int n1,n2;
  int b1,b2;

  n1=3;
  n2=5;
  b1=(n1>n2);
  b2=(n1<n2);

  printf("b1=%d b2=%d\n",b1,b2);

  return 0;
}
```

[14]関係演算子のオブジェクトは複素数型は取らない。

演習 4.17 test-bool.c のオブジェクト b1 と b2 を_Bool 型に置き換えたらどうなるか。

演習 4.18 演習 4.16 を test-bool.c を修正したプログラムで確かめよ。

4.7 算術演算子 — その 2

char 型、int 型、float 型、double 型などの算術型に対する、数学の 4 則演算に相当する算術演算子は 4.5 節 (p.33) で述べた。ここでは、次節で学ぶ for 文や while 文でしばしば用いられる複合代入演算子と増分減分演算子について説明する。

4.7.1 複合代入演算子

算術演算と代入演算を合わせて行う複合代入 (compound assignment) を行う演算子には次のものがある。オペランド (E1 と E2) は対応する算術演算子が許される型である。

	複合代入	単純代入
+=	E1 += E2	E1 = E1 + E2
-=	E1 -= E2	E1 = E1 - E2
*=	E1 *= E2	E1 = E1 * E2
/=	E1 /= E2	E1 = E1 / E2
%=	E1 %= E2	E1 = E1 % E2

E1 = E1 + E2 は数学の式と考えると奇妙だが、E1 の値に E2 の値を加えた値を左辺の E1 に代入するという意味である。

4.7.2 増分/減分演算子

増分演算子 (increment operator)++は、そのオペランドに 1 を加え、減分演算子 (decrement operator)--はそのオペランドから 1 を減ずる。

次の 3 つの演算は同じである。

- i = i+1;
- i += 1;
- ++i;

++と--は、前置演算子 (++i のようにオブジェクトの前に置く) とすることも後置演算子 (i++のようにオブジェクトの後に置く) とすることもできるが、++i は i の値を評価する前に増分し、i++は i の値を評価した後に増分する。

例 4.10 前置演算子++と後置演算子--違いは、次のプログラムで分かる。

```
/* increment.c */
#include <stdio.h>

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

  i = 5;
  j = i++; /* i の値を j に代入した後、 i を増分する */
  printf("i = %d j = %d\n",i,j);

  i = 5;
  k = ++i; /* i を増分した後、i の値を k に代入する */
  printf("i = %d k = %d\n",i,k);

  return 0;
}
```

j = i++; は、j に i の値 5 が代入されたあと、i が増分され 6 になる。これを k = ++i; とすると、i が増分され 6 になったあと、k に i の値 6 が代入される。

4.8 キャスト演算子

式の型は変えず、用いられたときに一時的に型を変えるキャスト演算子は、

```
(型) 式
```

と表す。キャスト演算子は、単項演算子 + や - と同じ優先順位である。

キャスト演算子を用いると、int 型データの平均値を double 型に取れる。

例 4.11 3 個の int 型数を入力し和と平均値を求める。

```
/****************************************
* cast.c                                *
* 3 個の int 型数を入力し和と平均値を求める。  *
* キャスト演算子を用いている。              *
****************************************/
#include <stdio.h>

int main(void)
{
  int a,b,c,sum;   /* sum を int 型と宣言している */
  double mean;

  printf("a=");
  scanf("%d",&a);
  printf("b=");
  scanf("%d",&b);
  printf("c=");
  scanf("%d",&c);
  sum = a+b+c;
  mean = (double) sum/3;   /* sum を double 型にキャストしている */

  printf("和は%d\n",sum);   /* 変換指定子は %d */
  printf("平均値は%f\n",mean);  /* 変換指定子は %f */

  return 0;
}
```