C言語でプログラミングを行うときのいろいろな約束事

C言語で使える演算子，関数，制御文の扱い

岸 哲夫

C言語で使える演算子とその優先順位は規格で決まっている．また，標準ライブラリ関数が用意されているが，プロトタイプ宣言をすると，そのプログラム中で使うことができる．ここでは，それらを使う上での注意点について解説する．（編集部）

C言語で使える演算子

C言語で使う演算子は，一般的な数式で使うものだけではありません．しかし基本的なルールさえ理解すれば問題なく対応できるはずです．

● 演算子の優先順位はあらかじめ決まっている

優先順位があいまいな場合はカッコで囲めば優先になることを覚えておくと便利です（**表1**）．

● 演算子の種類

C言語では比較的簡単に四則演算ができるように基本的な数式はそのまま使うことができます．

表1　演算子の結合規則と優先順位

種　類	演算子	結合規則	優先順位
式	() [] . -> ++ --	左 => 右	高
単項演算子	++ -- ! ~ + - * & sizeof	左 <= 右	
キャスト	(型)		
乗除余	* / %	左 => 右	
加減	+ -		
シフト	<< >>		
比較	< <= > >=		
等値	== !=		
論理積（ビット演算）	&		
排他的論理和（ビット演算）	^		
論理和（ビット演算）	|		
論理積	&&		
論理和	||		
条件	?:	左 <= 右	
代入	= += -= *= /= %= &= ^= |= <<= >>=		
コンマ	“ , ”	左 => 右	低

▶ =

= は左辺の変数に右辺の値を書き込みます．

▶ +

+ は左辺と右辺の値を加えた値を戻します．

▶ -

-は左辺から右辺の値を引いた値を戻します．

▶ /

/は左辺を右辺で割った商を戻します（左右が整数ならば商も整数）．

▶ %

% は左辺を右辺で割った余りを戻します（左右は整数）．

リスト1に四則演算・前置き/後ろ置きを動作確認するソース・リストを，**図1**に四則演算・前置き/後ろ置きを動作確認して表示した結果を示します．

● ビット演算子 ── 下位ビット，上位ビット，またはビット・マスクが簡単にできる

ビット演算子を使って，**リスト2**のプログラム・ソースで論理積，排他的論理和および論理和の動作確認を行ってみます．**図2**はコンパイル実行した結果です．

このように論理積を使うと下位ビット，上位ビット，またはビット・マスクが簡単にできることを理解してください．排他的論理和，論理和の演算内容も理解してください．

● シフト演算子 ── 下位ビットを丸めるときなどに使う

リスト3にシフト演算子を実証するソース・リストを，**図3**にシフト演算子を実証して表示した結果を示します．

10101100の下3ビットがクリアされて10101000になったことがわかると思います．

● キャスト演算子 ── 明示的型変換をする

明示的型変換をすることをキャストすると言います．その演算子が「キャスト演算子」です．もちろん不可能な型変換もありますが，int を float に変換することは問題ありません．ただし，逆の場合では小数部が落ちてしまうので，注意してくだ

KEYWORD ── 演算子，=，+，−，/，%，ビット演算子，シフト演算子，キャスト演算子，sizeof 演算子，関数，プロトタイプ宣言，protoize，unprotoize，typeof，lvalue，aligned，mode，nocommon，packed，section，transparent_union，unused，if 文，switch 文，for 文，while 文

リスト1　四則演算・前置き/後ろ置きを動作確認するソース・リスト(test34.c)

```
/*
 *演算子の基本
 */
#include<stdio.h>

int main(void)
{
        int a=10;
        int  b=3;
        int  c;

//四則演算
        printf("a=%dIn",a);
        printf("b=%d¥n",b);
        printf(" a+b :%d¥n",a+b);
        printf(" a-b :%d¥n",a-b);
        printf(" a*b :%d¥n",a*b);
        printf(" a/b :%d¥n",a/b);
//文字％を書く場合は％％と書く
        printf(" a%%b :%d¥n",a%b);
        printf(" c=a+b :%d¥n",c=a+b);
        printf("c=%d¥n¥n",c);

//前置か後置か
        printf("a=%d¥n",a);
        printf(" a++ :%d¥n",a++);
        printf("a=%d¥n",a);
        printf(" ++a :%d¥n",++a);
        printf("a=%d¥n",a);

        return 0;
}
```

```
$ ./test34
a=10
b=3
 a+b :13
 a-b :7
 a*b :30
 a/b :3
 a%b :1
 c=a+b :13
c=13

a=10
 a++ :10
a=11
 ++a :12
a=12
```

図1　四則演算・前置き/後ろ置きを動作確認して表示した結果

さい．なお，有効桁数は変換する側にあわせて縮小されます．

リスト4にキャスト演算子の動作確認を行うソース・リストを，**図4**にキャスト演算子の動作確認を表示した結果を示します．

● 条件演算子 ―― 可読性を考えて使う

C言語では，何かのプログラムを書くときに，いくつもの書き方が許される場合があります．たとえば，

```
if(式1)
    {
       式2
    }
else
    {
        式3
    }
```

上のような条件式を省略して，次のように記すことができます．

```
式1 ? 式2 : 式3
```

筆者は，後者の書き方は後でソースを見たときにわかりにくいと感じるので，あまり使いません．昔からのプログラマで，

リスト2　ビット演算子を動作確認するソース・リスト(test35.c)

```
/*
 *bit演算子の基本
 */
#include<stdio.h>

int main(void)
{
    char data =    0xff;

//
    printf("%X¥n",data & 0xf0);        //上位ビットを取り出す
    printf("%0.2X¥n",data & 0x0f);     //下位ビットを取り出す
    printf("%X¥n",data & 0x10);
    printf("%X¥n",data & 0x1f);
//
    data =    0x11;

//
    printf("%X¥n",data & 0xf0);        //上位ビットを取り出す
    printf("%0.2X¥n",data & 0x0f);     //下位ビットを取り出す
    printf("%X¥n",data & 0x10);
    printf("%X¥n",data & 0x1f);
//
    data =    0x01;
    printf("%X¥n",data & 0x01);  //上位1ビット目を取り出す  1→1
    data =    0x02;
    printf("%X¥n",data & 0x01);  //上位1ビット目を取り出す  0→0
    data =    0x08;
    printf("%X¥n",data & 0x08);  //上位4ビット目を取り出す  1→8
    data =    0x04;
    printf("%X¥n",data & 0x08);  //上位4ビット目を取り出す  0→0
    data =    0x04;
    printf("%X¥n",data | 0x08);
                          //00000100 | 00001000  C→00001100
    data =    0x0c;
    printf("%X¥n",data ^ 0x08);
                          //00001100 ^ 00001000  4→00000100

    return 0;
}
```

```
$ ./test35
F0
0F
10
1F
10
01
10
11
1
0
8
0
C
4
```

図2　ビット演算子を動作確認して表示した結果

リスト3　シフト演算子を動作確認するソース・リスト(test36.c)

```
/*
 *シフト演算子の基本
 */
#include<stdio.h>

int main(void)
{
        unsigned  char data =    172;      //10101100
        unsigned  char res;

//True Color(8bits256階調)からHigh Color(5bits32階調)に
                                                    階調を落とす
        res  =    data >> 3;
        data =    res  <<   3;
        printf("%X¥n",data);               //下位3ビットを捨てる
//
    return 0;
}
```

```
$ ./test36
A8
```

図3　シフト演算子を動作確認して表示した結果