C 言語の学習
型変換・記憶クラス・初期化・演算子

山本昌志*

2004 年 5 月 12 日

1　本日の学習内容

C 言語の基本的な部分を学習する。内容は、教科書の 5〜8 章である。各章の内容と理解すべきことは、以下の通りである。

5 章　型変換

- 暗黙の型変換により、データは自動的に無難な型に変換される。
- 任意の型に変換するためには、明示的な型変換 (キャスト) を使う。

6 章　記憶クラス

- ローカル変数とグローバル変数の違い。
- 自動変数 (auto) と静的変数 (static) の違い。

7 章　初期化

- 配列の宣言と同時に代入演算子 (=) を用いて、変数に値が代入できる。

8 章　演算子

- 関係演算子、等価演算子、論理演算子の使い方。
- インクリメント、デクリメント演算子の使い方。
- 代入演算子の使い方。

2　型変換 (教科書の 5 章)

メモリーに格納されているビットの並びを考えると、コンピューターでは同じ型の変数同士で演算を行うのが望ましい。プログラマーはそのようにソースコードを書くべきであるが、避けられないこともある。そのようなときに、暗黙の型変換、あるいは明示的な型変換 (キャスト) が使われる。

*独立行政法人　秋田工業高等専門学校　電気工学科

2.1 暗黙の型変換 (p.62)

教科書には代入時型変換・関数の引数型変換・単項型変換・算術変換が書かれているが、諸君にとって重要なのは、最初と最後の型変換である。暗黙の型変換は、いろいろとルールが書かれているが、精度の高い方に変換され、プログラマーにとって都合の良い仕様なので、あまり気にする必要はない。唯一、整数と整数の除算のみ気を付ければよい。C 言語では、整数同士の除算の結果は整数となる。これについては、後の練習問題で体験してもらう。

2.1.1 代入時型変換 (p.62)

代入演算子 (=) は、右辺の変数の値を、左辺の変数に代入する。右辺と左辺の型が異なる場合に、型変換が行われる。リスト 1 をみて、動作の内容を理解して欲しい。

9 行 倍精度実数型 (double) の値を整数型 (int) の変数へ代入

10 行 整数型 (int) の値を倍精度実数型 (double) の変数へ代入

12 行 変数 j を 10 進数 (%d) で、y の値を浮動小数点数 (%f) で表示 (教科書 p.322 変換指定子)。これらの間に、タブ (\t) で適当な空白を入れている (教科書 p.28 表 2-4)。

リスト 1: 代入時型変換の例

```
1  #include <stdio.h>
2  int main(){
3    int i,j;
4    double x,y;
5
6    i=123;
7    x=4.567;
8
9    j=x;
10   y=i;
11
12   printf("j = %d\tj = %f\n", j, y);
13
14   return 0;
15 }
```

実行結果

```
j = 4    j = 123.000000
```

リスト 1 の結果について、以下を考えよ。

[**練習 1**] 代入時型変換が行われている行を示せ。また、代入時型変換が行われていない行を示せ。

[**練習 2**] 実行結果がなぜそのようになったか考えよ。

2.1.2 算術変換 (p.64)

コンピューター内部で算術演算の処理を行う場合、それは同じ型の方が都合がよい。同じ性質のビット列の方が都合が良いことは明らかである。そのため、演算を行う 2 つ型が異なる場合、どちらかに統一しなくてはならない。C 言語では、表現能力の高い型へ統一されて演算が行われることになっている。

倍精度実数と整数の演算を行う場合、それは倍精度実数で計算されるので、プログラマーは気にしなくて良いのである。反対に、整数型に統一されると、桁落ちにより計算精度が著しく低下する。これを避けるように C 言語の仕様は決まっている。

2.2 明示的な型変換 (キャスト)

データの型を変更したい場合に明示的な型変換 (キャスト) を使う。これを使うことにより、倍精度実数型のデータを整数型に、あるいはその反対など、プログラマーのお望みの型に変換できる。例えば、整数型のデータ `i` と `j` の除算などに便利である。`i=3, j=4` として、`i/j` を計算すると `0` になってしまいプログラマーの意図したとおりに動作しない。このときに、`(doubel)i` として、整数型の変数の値を一時的に倍精度実数にして計算すると問題が解決される。

9 行 整数変数 `i` の値を一時的に、倍精度実数に変換している。そうすると、倍精度実数と整数の除算になる。次に、暗黙の型変換が適用され、最終的には倍精度実数同士の除算になり、倍精度実数の演算結果が得られる。

リスト 2: キャストを使用した例

```
1  #include <stdio.h>
2  int main(){
3    int i,j;
4    double x;
5
6    i=3;
7    j=4;
8
9    x=(double)i/j;
10
11   printf("x = %f\n", x);
12
13   return 0;
14 }
```

実行結果

```
x = 0.750000
```

以下の練習問題を実施せよ。

[練習 1] リスト 2 を書き換えて、以下の結果を調べよ。そして、その理由を考えよ。

```
x=i/j                    x=i/4.              x=i*1.0/j
x=(double)(i/j)          i/(double)j
```

3　記憶クラス (教科書の 6 章)

記憶クラスの話は、関数 (サブルーチン) を使わないと御利益がない。そこで、本日はこのプリントを読む程度にとどめるのが良いだろう。関数の学習の時に、ちゃんと説明する。

3.1　ローカル変数とグローバル変数 (p.68)

変数には宣言をする場所によりローカル変数とグローバル変数がある。

ローカル変数　関数の外で宣言され、その関数の中だけで使用できる。関数がコールされるとメモリー上に変数が配置される。その関数の処理が終わるとその変数は消滅する。通常、よく使われる。

グローバル変数　関数の外で宣言され、どの関数でも使用できる。プログラムが起動されるとメモリー上に変数が配置される。プログラムが終了するまで、変数は維持される。

教科書の p.69 の図 6-1 を見て欲しい。ここでは、こんなものがあると思うだけでよい。関数の時にもう少し分かりやすく説明する。ただ、グローバル変数はできるだけ使わない方が良い。プログラムの独立性が低くなるし、非常に分かりづらいバグが発生することがある。この意味については、もう少しプログラムに馴れれば理解できるであろう。

この授業で諸君は、グローバル変数を使うプログラムを書くことはほとんどないであろう。

3.2　自動変数 (auto) と静的変数 (static)　(p.70, p.77)

静的変数は、変数宣言の前に `static` と付ければ良い。一方、今まで学習してきた変数は `static` が無いので、自動変数である。それらの違いは、次に通りである。

自動変数　関数内でのみ値を保持する。関数の動作が終わると、メモリーの解放され、その値は 2 度と使えない。新たにその関数をコールすると、新たにメモリーを確保する。この場合、前の場所と同じとは限らない。

静的変数　プログラムが起動されたときにメモリーが確保され、プログラムが終了するまでそれが維持される。

この授業で諸君は、グローバル変数を使うプログラムを書くことはほとんどないであろう。

4　初期化 (教科書の 7 章)

4.1　暗黙の初期化 (p.90)

変数宣言をしただけで値の設定を行わない場合、暗黙の初期が行われる。変数宣言をすると、コンピューターのメモリーが予約される。予約されたメモリーの各ビットは、`0` か `1` が詰まっているわけでなにがしかの値がある。この値であるが、変数の記憶クラスによって異なる。

- 自動変数とレジスタ変数の場合、値がどのようになっているか分からない。
- 静的変数 (static) とグローバル変数の場合、0 となっている。

自動変数の値の設定を行わないで、ゼロになっていると勘違いし、プログラムを作成しすることがある。自動変数を使うときには十分注意が必要である。

[**練習 1**] 整数型の変数として、グローバル変数 i 、ローカルの自動変数 j 、ローカルの静的変数 k を宣言して、それに格納されている値を調べよ。(ヒント:p.90 の上の方のプログラム)

4.2 変数の初期化 (p.91)

教科書の p.91 に書かれているように、変数宣言とともに代入演算子を用いて、初期値を設定できる。この設定する初期値は、コンパイル時に値が計算できなくてはならない。静的変数 (static) はコンパイル時に値が決定されて、それがメモリーの初期値となる。自動変数はその関数が実行されるときに初めて、メモリーが確保されて、その値が設定される。この辺の話は、関数の時にするので、あまり気にしないで欲しい。

5 演算子 (教科書の 8 章)

5.1 算術演算子 (p.107)

算術演算子 (教科書 p.107) については、今更説明するまでもないであろう。この中で、剰余[1](%) はかなり便利である。11%4 の演算結果は、3 である。覚えておくと良い。

5.2 関係演算子、等価演算子、論理演算子 (p.108〜)

これらの演算は、主に論理演算に使われる[2]。論理が正しい (真 True) か誤り (偽 False) という演算である。演算の結果は、真か偽のいずれかである。真の場合が 1 で、偽の場合が 0 である。

5.2.1 関係演算子 (p.108)

算術演算子の + は 2 つのデータの加算を行い、その和を返す。5+8 が 13 になるようにである。関係演算子 (p.108) も同じで、2 つのデータの演算を行い値を返す。関係演算子が返す値は、0 か 1 である。たとえば、10<20 の演算結果は 1 、20>10 は 0 になる。もちろん、演算を行う 2 つの数値は、実数でも良い。関係演算子は、大小の比較を行ってその判定をしていると考える。難しいことは、なにもない。

[**練習 1**] 以下に示すそれぞれの a の値を計算し、結果を表示するプログラムを作成せよ。4 番目の

[1] 教科書では余りと表現している
[2] 論理演算は、制御文 if とともに使われることがほとんどである。

演算結果については、演算子の優先順位 (p.135 表 8-3) が問題となる。

```
a=1+2          a=1<2          a=2>1
a=1+3>=2+2     a=5*((1<2)+(2<4))
```

[**練習 1**] 次の d の値を C 言語のプログラムで計算せよ。なぜ d の値がそのようになるのか考えよ。ヒント、教科書 p.34 表 3-1 を見よ。

```
a=1122334455;          b=1122334455;
c=a+b;                 d=1<c;
```

5.2.2 等価演算子 (p.108)

関係演算子が大小の比較を表すのに対して、等価演算子は等しいか否かを表す。

[**練習 1**] 教科書の p.108 の等価演算子の表を見ながら、以下の演算結果の値を考えよ。もし分からない場合は、プログラムを作成して、計算して見よ。

```
100 == 100     3 == 5     3.0 == 3
6 != 5         5 != 5
```

5.2.3 論理演算子 (p.109)

論理演算子は 2 年生の時に学習したブール代数の演算子である。ブール演算では、否定は NOT で、論理積は AND で、論理和は OR で表す。しかし、p.109 の表に示すような記号を用いる。当然これも真理値表で書くことができて、表 1〜3 のように表す。

演算の対象が 0 の場合は偽 (0) として扱われ、1　の場合は真 (1) となる。これは簡単でブール代数の演算そのものである。表 1〜3 のようになる。

表 1: 否定の演算

a	!a
0	1
1	0

表 2: 論理積の演算

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

表 3: 論理和の演算

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

問題は、演算の対象が 0 や 1 以外の場合である。プログラマーからすれば、コンパイラーがエラーを出すか、実行時にエラーを出して止まってくれれば良いのだが、実際にはそうはならない。C 言語の仕様では 0 と 1 以外の場合、それは真 (1) として扱うと決まっている。C 言語では、

- 0 は偽
- 0 以外は真

として取り扱われると覚えておく。そうすると、論理演算は表 4〜6 のようになる。

表 4: C 言語の否定演算

a	!a
0	1
0 以外	0

表 5: C 言語の論理積

a	b	a && b
0	0	0
0	0 以外	0
0 以外	0	0
0 以外	0 以外	1

表 6: C 言語の論理和

a	b	a || b
0	0	0
0	0 以外	1
0 以外	0	1
0 以外	0 以外	1

5.3 インクリメント、デクリメント演算子 (p.110)

インクリメント演算子 (++) は 1 加算し、デクリメント演算子 (--) は 1 減算する演算子である。教科書に書いてあるように、a=a+1 あるいは a=a-1 の代わりに使われる。カウンターとして使っている変数の値を変化させるときに、使われることが多く、代入演算子 (=) を使うよりも、インクリメントやデクリメント演算子を使う方が C 言語風で格好良いのである。

リスト 3: インクリメントとデクリメントの例

```
1  #include <stdio.h>
2
3  int main(){
4    int i,j;
5
6    i=10;
7    j=10;
8
9    i++;
10   j--;
11
12   printf("i=%d  j=%d\n",i,j);
13
14   return 0;
15 }
```

[練習 1] リスト 3 の動作を確かめよ。

5.4 代入演算子 (p.118)

単純代入演算子 (=) は説明しなくても分かっていると思うだろうが、これがどうしてなかなかちゃんと理解されていないのである。単純代入演算子 (=) は数学のイコール (=) と異なり、これは演算子である。演算

子と言うことであるから、これを挟んだ変数に対して操作をする[3]。 その操作は、左辺の式の値を右辺の変数に代入する。必ず、右辺は式[4]で、右辺は変数でなくてはならない。もし、左辺と右辺が等しいか否かの比較は等価演算子 (==) を使う。C 言語では、代入演算子 (=) と等価演算子 (==) はしっかり区別を付けなくてはならない。

複合代入演算子もよく使われる。特に += は使われることが多いので、よく覚えておかなくてはならない。a の変数の値に b を加算する場合、a=a+b とすればよいが、C 言語では a+=b と書くのが普通である。前者でも問題なく実行できるが、後者の方が C 言語風で格好良いとされている。ほかの複合代入演算子も同じである。

リスト 4: 複合代入演算子の例

```
1  #include <stdio.h>
2
3  int main(){
4    int i,j;
5
6    i=3;
7    j=6;
8
9    i+=j;
10
11   printf("i=%d   j=%d\n",i,j);
12
13   return 0;
14 }
```

[**練習 1**] リスト 4 の動作の結果を考えよ。

[3]数学の 3+5 = 8 の意味は、演算子 + が 3 と 5 に作用してその結果は、8 に等しい。+ は演算子であるが、= は演算子ではない。
[4]定数や変数が一つの場合も、式の範疇である。