4.3 種々の代入演算子

- 代入文

```
c = c + 3;
```

  は，加算代入演算子を使って

```
c += 3;
```

  と書ける
- 一般に

  変数 = 変数 operator 式;

  を

  変数 operator = 式;

  にできる
- 他の代入演算子の使用例:

```
d -= 4  (d = d - 4)
e *= 5  (e = e * 5)
f /= 3  (f = f / 3)
g %= 9  (g = g % 9)
```

4.4 インクリメント演算子とデクリメント演算子

- インクリメント (increment) 演算子:
  - 変数の内容を 1 増やす
  - `++c` または `c++`　(`c += 1`)
- デクリメント (decrement) 演算子:
  - 変数の内容を 1 減らす
  - `--c` または `c--`　(`c -= 1`)
- プリインクリメント (preincrement)
  - 変数の 前に `++` や `--` を書く
  - 変数を式の中で使う 前に 1 増やす (減らす)
- ポストインクリメント (postincrement)
  - 変数の 後ろに `++` や `--` を書く
  - 変数を式の中で使った 後で 1 増やす (減らす)
- 変数 c の値が 5 のとき
  - `printf("%d", ++c);` は 6 を表示
  - `printf("%d", c++);` は 5 を表示
  - 両方とも，文を実行した後の c の値は 6
- 式の中で使われない場合には，プリとポストの効果は同じ。
  -

```
++c;
printf("%d", c);
```

- `c++;`
  `printf("%d", c);`

- よいプログラミング作法
  - 演算子のオペランドを評価する順序は，ANSI 規格で規定されていない (処理系依存)
  - `++c / ++c` や `printf("%d %d\n", ++c, ++c)` と書くべからず (どちらの ++c が先に実行されるかがあいまい)

4.5 for 反復構造

4.5.1 カウンタ制御による反復の復習と補足

- カウンタ制御による反復には次の項目が必要
  1. 制御変数 (ループカウンタ) の名前 (変数宣言)
  2. 制御変数の初期値
  3. 各ループ毎に行う制御変数のインクリメント (デクリメント)
  4. 制御変数が最終値か否かをテストする反復条件 (ループ継続条件)
- 例: 1 から 10 までの整数を表示する

```
int counter = 1;            /* 制御変数の宣言と初期化 */
while (counter <= 10) {     /* 反復条件 */
   printf("%d\n", counter);
   ++counter;               /* インクリメント */
}
```

- 上のコードは次のように書いてよい — 実行を伴う式を「(条件式)」に書ける

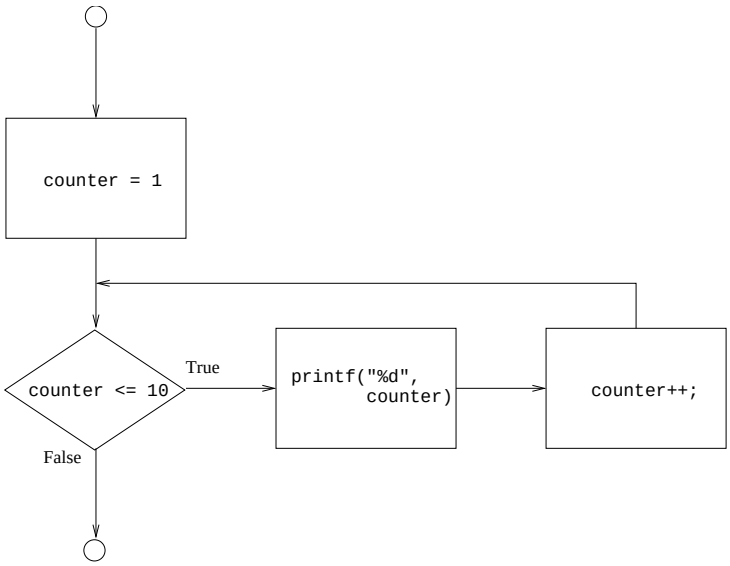
```
int counter = 0;            /* 制御変数の宣言と初期化 */
while (++counter <= 10)     /* インクリメントと反復条件 */
   printf("%d\n", counter);
```

4.5.2 for 反復構造

- for 反復構造 (for ループ): カウンタ制御をより簡単に表現できる
- 例: 1 から 10 までの整数を表示する

```
for (counter = 1; counter <= 10; counter++)
   printf("%d\n", counter);
```

- for 反復のフローチャート例



- for ループの書式

```
for (式 1; 式 2; 式 3)
   文
```

式 1: 初期化, 式 2: 反復条件, 式 3: インクリメント (カウンタ増加)

- for ループと等価な while ループ

```
式 1;
while (式 2) {
   文
   式 3;
}
```

- 参考: コンマ演算子 — for の式 1 と式 3 にはコンマ区切りの式の並びを書ける[*7]
  - 例:

```
for (i = 0, j = 0; j + i <= 10; j++, i++)
   printf("%d\n", j + i);
```

  - コンマ (,) は優先順位最低の演算子
  - 演算結果は最右式の値となる
- for 構造の 式 1, 式 2, 式 3 は省略可能

```
for ( ; ; )
  文
```

は「文」を無限に繰り返す = 無限ループ

- よくあるプログラミングエラー

```
for (counter = 1; counter <= 10; counter++);
   printf("%d\n", counter);
```

[*7] 多用は勧めない

4.5.3 for 構造: 補足と注意

1. for 構造の初期設定部，ループ継続条件部，増減部に算術式を含めることができる。
   x が 2， y が 10 のとき，
   ```
   for (j = x; j <= 4 * x * y; j += y / x)
   ```
   と
   ```
   for (j = 2; j <= 80; j += 5)
   ```
   は等価
2. for の式 3 (インクリメント; 増分) は負になり得る = デクリメント
3. ループ継続条件 (式 2) が最初から偽の場合
   - for 構造の本体は一度も実行されない
   - その直後にある文に制御が移る
4. 制御変数は，しばしば for 構造の本体の中で出力や計算に使われるが，それは必須ではない。

4.5.4 for 構造の使用例

■リスト 4.2 **for を使った合計**

```
 /* for 文による合計（2 から 100 までの偶数の和を求める）*/
#include <stdio.h>

int main()
{
  int sum = 0, number;

  for (number = 2; number <= 100; number += 2)
    sum += number;

  printf("Sum is %d\n", sum);

  return 0;
}
```

■リスト 4.3 **合格者と不合格者を数える**

```
/* count number of passes and failures */
#include <stdio.h>

int main()
{
     int score, i;
```

```
    int npass = 0, nfail = 0;

    printf("Enter five scores: ");

    for (i = 0; i < 5; i++) {
        scanf("%d", &score);

        if (score >= 60)
            ++npass;
        else
            ++nfail;
    }

    printf("passed %d\n", npass);
    printf("failed %d\n", nfail);

    return 0;
}
```

実行結果

```
Enter five scores: 100 50 90 85 60
passed 4
failed 1
```

■リスト 4.4　**整数のべき乗を求める**

```
/* power calculation (m^n)
   assume that m and n is integer and n is larger than 0 */
#include <stdio.h>

int main()
{
    int m, n;
    int power = 1, i;

    printf("Enter m and n: ");
    scanf("%d%d", &m, &n);

    for (i = 0; i < n; i++)
        power *= m;
```

```
    printf("%d^%d = %d\n", m, n, power);

    return 0;
}
```

実行結果

```
Enter m and n: 5 3
5^3 = 125
```

■リスト 4.5 for **を用いた二重ループ** (**表の出力**)

```
/* double loop */
#include <stdio.h>

int main()
{
    int i, j;

    for (i = 0; i < 3; i++) {
        for (j = 0; j < 5; j++) {
            printf("(%d,%d) ", i, j);
        }
        printf("\n");
    }

    return 0;
}
```

実行結果

```
(0,0) (0,1) (0,2) (0,3) (0,4)
(1,0) (1,1) (1,2) (1,3) (1,4)
(2,0) (2,1) (2,2) (2,3) (2,4)
```