復習

# いろいろな変数型(2)

**char**

1バイト　→　英数字1文字を入れるのにぴったり
アスキーコード　→　文字と文字列で学習

**int**

4バイト　もっとも標準的な整数型

**float**

2進法で10進実数を表わすので誤差がある
（有効数字8桁程度）

4バイト　単精度実数型(単精度浮動小数点型)

**double**

floatより高精度(有効数字15桁程度)

8バイト　倍精度実数型(倍精度浮動小数点型)

復習

# 計算機における有効桁数と計算精度

## 有効数字

数学　$3 \times \frac{1}{3} = 1$

計算機　$3 \times 0.33333333 = 0.99999999$

有効数字（有効桁数8桁）

## ２進法による小数点以下の数値の表現

２進数　１１１１．１１１１　＝ 15.9375

$1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$
$= 8 + 4 + 2 + 1$
$= 15$

$1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} + 1\times2^{-4}$
$= 0.5 + 0.25 + 0.125 + 0.0625$
$= 0.9375$

# 少数以下の2進数による10進数値の表現

復習

$(100)_2 = 1\times2^2 = 4$
$(10)_2 = 1\times2^1 = 2$
$(1)_2 = 1\times2^0 = 1$
$(0.1)_2 = 1\times2^{-1} = 0.5$
$(0.01)_2 = 1\times2^{-2} = 0.25$
$(0.001)_2 = 1\times2^{-3} = 0.125$
$(0.0001)_2 = 1\times2^{-3} = 0.0625$

少数以下の10進数値には，有限桁数の２進法では表せない数値がある．

丸め誤差

問題：10進数の 0.1 は2進数で正確に表せるか？

$(0.001)_2 = 1\times2^{-3} = 0.125$ 大きすぎ！
$(0.0001)_2 = 1\times2^{-4} = 0.0625$ 小さすぎ！
$(0.00011)_2 = 1\times2^{-4} + 1\times2^{-5} = 0.0625 + 0.03125 = 0.09375$ まだ小さい！
$(0.000111)_2 = 1\times2^{-4} + 1\times2^{-5} + 1\times2^{-6}$
$= 0.0625 + 0.03125 + 0.015625 = 0.109075$ 大きすぎ！
$(0.0001101)_2 = 1\times2^{-4} + 1\times2^{-5} + 1\times2^{-7}$
$= 0.0625 + 0.03125 + 0.0078125 = 0.1015625$ まだ大きい！
$(0.00011001)_2 = 1\times2^{-4} + 1\times2^{-5} + 1\times2^{-8}$
$= 0.0625 + 0.03125 + 0.00390625 = 0.09765625$ 惜しい！

復習

# 定数値でよくある間違い

```
int aa, bb, cc = 1;
aa = 1/3*30;
bb = cc/3*30;
printf("aa = %d    bb = %d¥n", aa, bb);
```

1 / 3 * 30
= 0 * 30
= 0

整数同士の演算なので中間結果も整数値
1÷3 = 0.333... = 0

どんな結果になる？

```
aa = 0    bb = 0
```

プログラムの世界では...
1.0　浮動小数点数(double型)
1　整数

```
int aa;
aa = 1.0/3*30;
printf("aa = %d", aa);
```

1.0 / 3 * 30
= 0.333... * 30
= 10

```
aa = 10
```

浮動小数点数と整数の演算結果はdouble型

復習

# いろいろな演算子(1)

## 代入演算子

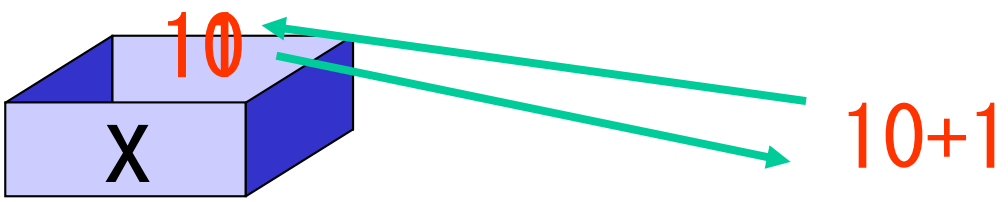
'='は等号ではなく代入の意味

```
x = 10;
x = x + 1;
```

x の解は不定？？？
= は ← の意味

```
x = x + 1;
```
⟶ x ← x + 1

変数(箱) xから値を取り出して，1を足して変数xに代入する



# いろいろな演算子(2)

復習

## 算術演算子

| | |
|---|---|
| + | 足し算 |
| − | 引き算 |
| * | 掛け算 |
| / | 割り算 |
| % | 剰余算（例）x = 5 % 3 ⇒ x = 2 |

割り算の余り
5÷3 = 1 … 2

| | |
|---|---|
| ++ | インクリメント演算子<br>aa++ は aa = aa+1 の省略形 → aaの値を1増やす |
| −− | デクリメント演算子<br>aa−− は aa = aa−1 の省略形 → aaの値を1減らす |

```
aa = 10;
aa++;          /* この文を実行した後のaaの値は11になる */
```

aa = aa++; と書くのは誤り！

理由
式中の aa++ を aa = aa+1 で書き換えると aa = aa = aa+1

実は，インクリメント演算子の性質のため，aa=aa++;と書いても正しい結果が得られる．その理由が説明できない人は，この書き方は間違いだと覚えておこう！

復習

# scanf()関数による数値の入力

```
int x;
printf("値を入力してください. ");
scanf("%d", &x);
printf("入力された値は, %dです. ¥n", x);
```

```
値を入力してください. 15
入力された値は, 15です.
```

キーボードから入力

```
int aa;
scanf(" %d ", & aa );
```

整数値の入力では%dとする.

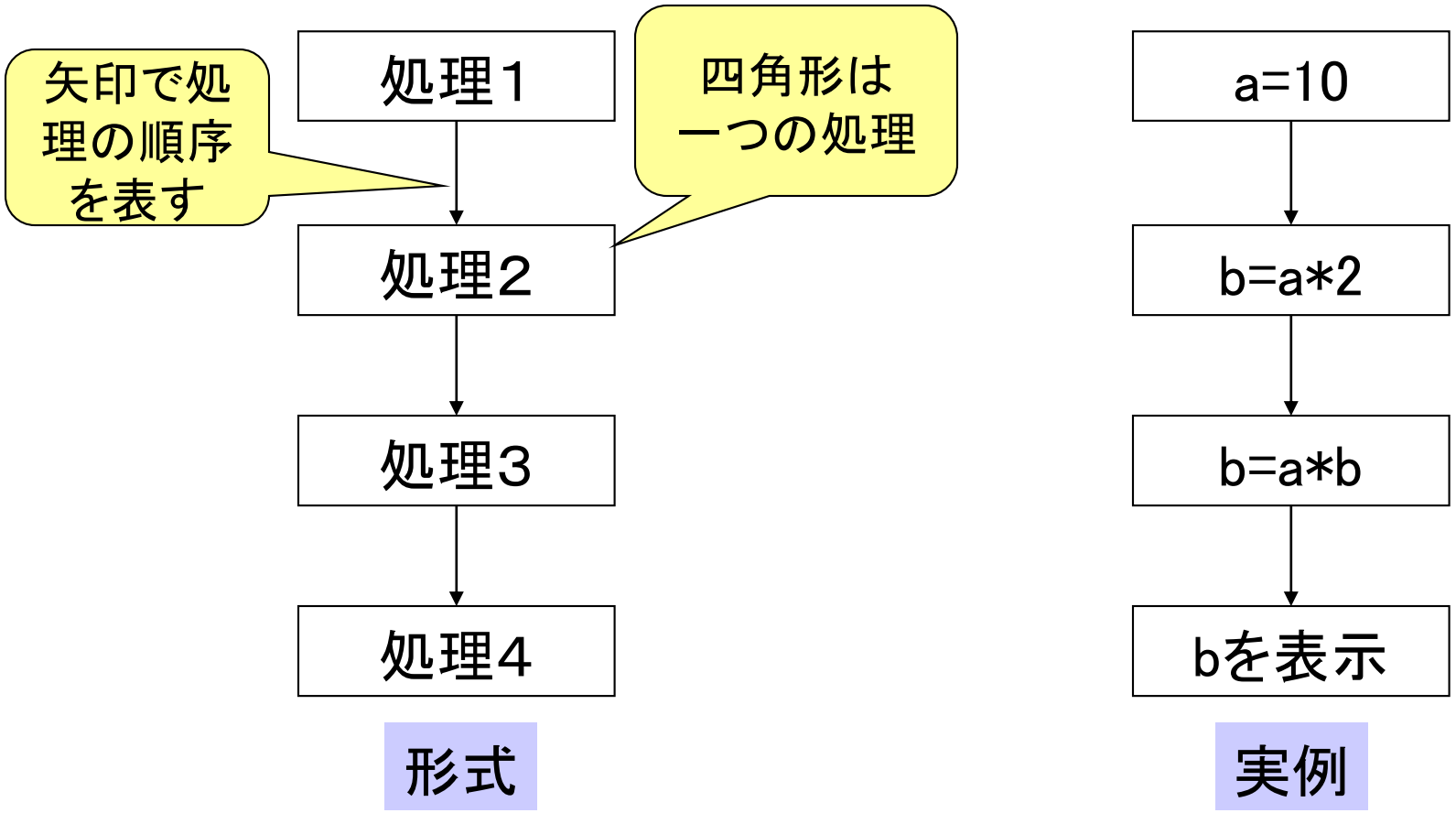
変数名の前には&をつける

```
float bb;
scanf(" %f ", & bb );
```

実数値の入力では%fとする.

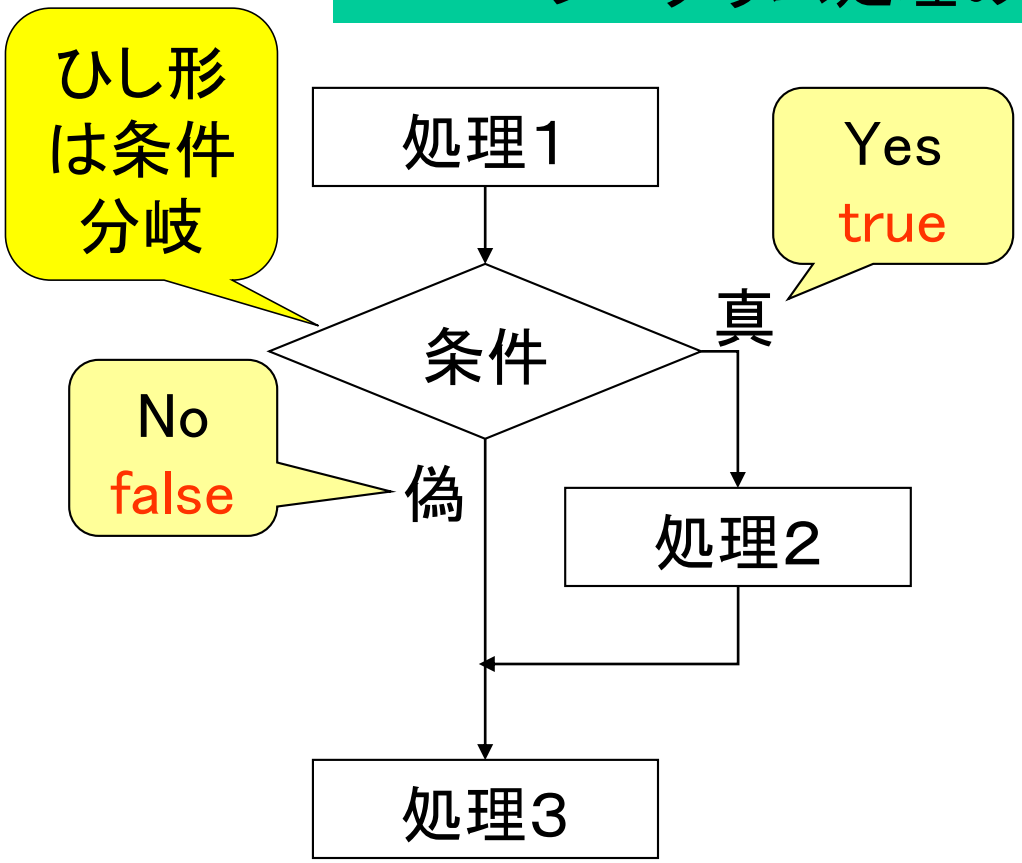
# フローチャート(1)

フローチャート→
プログラム処理の流れを示す図

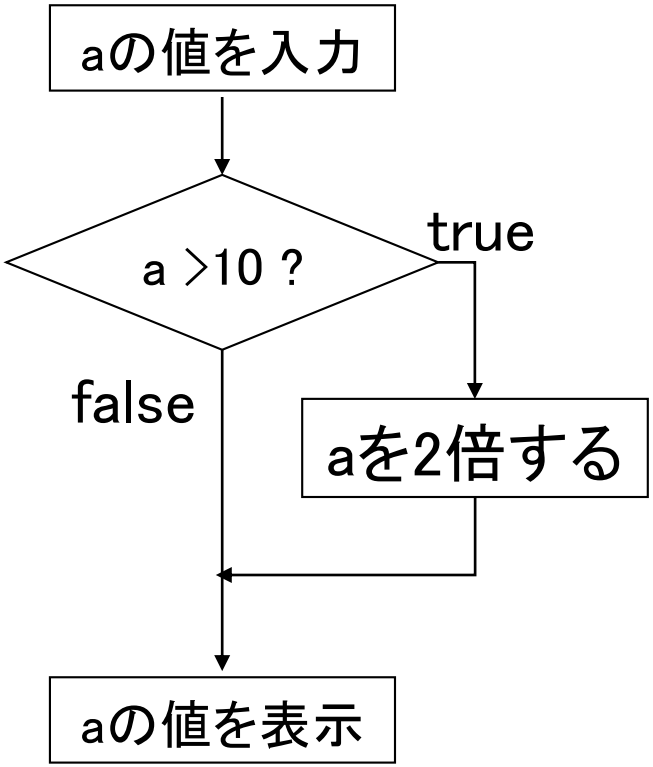


# フローチャート(2)

条件分岐→
　　プログラム処理の流れを変える

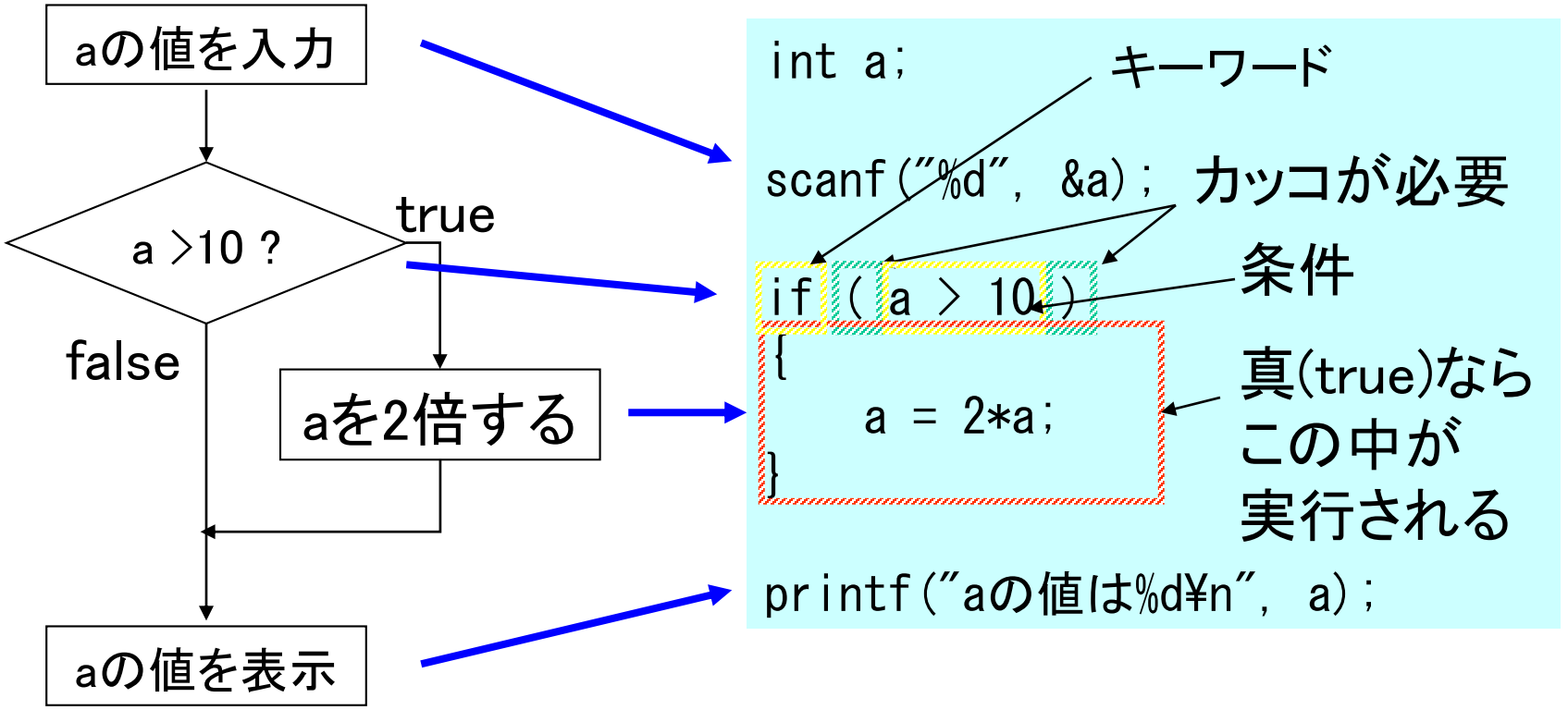



形式

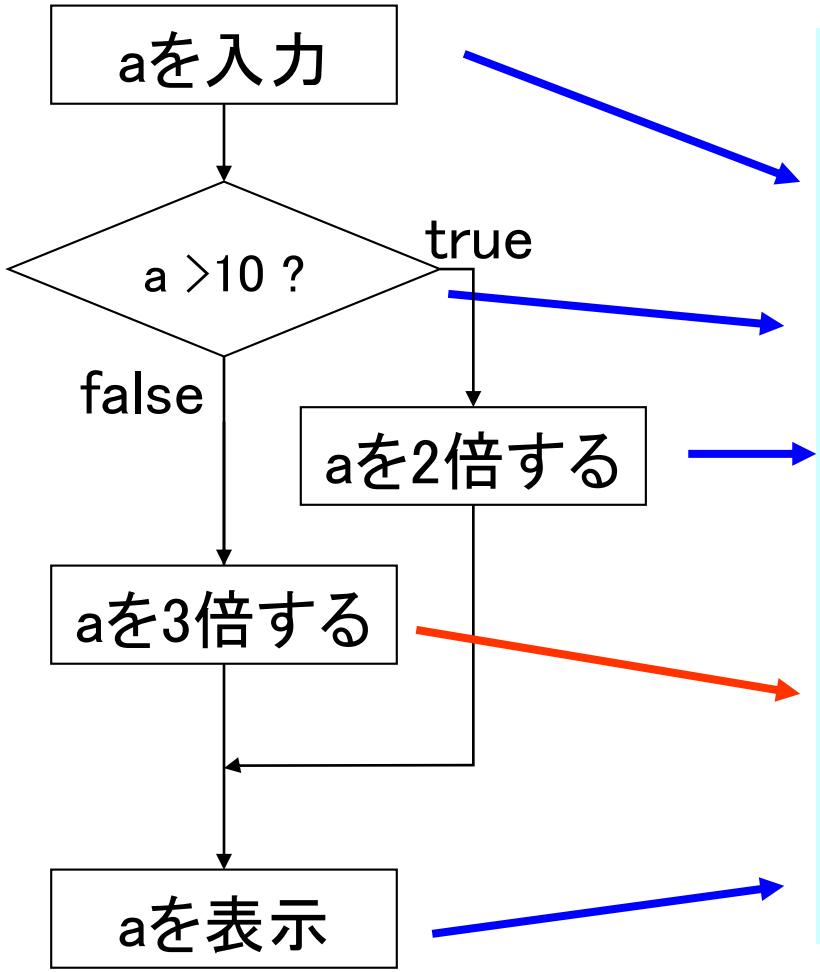



実例

# if ～ 選択制御文(条件分岐，if文)



# if ～ else～選択制御文(条件分岐，if～else～文)




```
int a;

scanf("%d", &a);

if ( a > 10 )
{
    a = 2*a;
}
else
{
    a = 3*a;
}

printf("aの値は%d¥n", a);
```

真(true)なら
この中が
実行される

偽(false)なら
この中が
実行される

# いろいろな条件式と関係演算子

| | Cのソース |
|---|---|
| a ＞ b | if (a > b) |
| a ≧ b | if (a >= b) |
| a ＜ b | if (a < b) |
| a ≦ b | if (a <= b) |
| a ＝ b | if (a == b) |
| a ≠ b | if (a != b) |
| 5 ＜ a かつ a ＜ 10 | if (5<a && a<10) |
| 10 ＞ a または 15 ＜ a | if (10>a \|\| 15<a) |
| 10 ＞ a ではない | if ( !(10 > a) ) |

大原則
「=」記号は常に右側に付く

半角文字

！注意！
等号が二つ

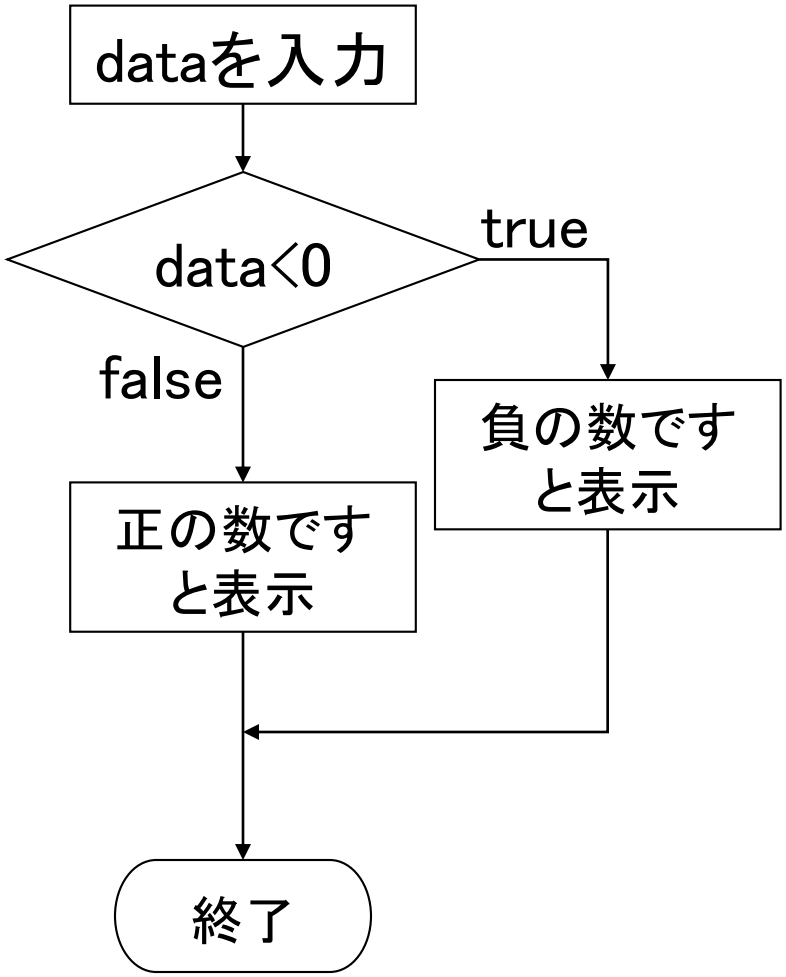
かつ

これはダメ
5 ＜ a ＜ 10

または

否定

# if～else～文の例




```
#include <stdio.h>
int main(void)
{
  int data;

  scanf("%d", &data);
  if ( data < 0 )
      printf("負の数です¥n");
  else
      printf("正の数です¥n");
}
```

カッコ{ }
が無い！

# 単文と複文

文は命令(関数)や式

C言語の文は単文か複文のどちらか

## 単文

```
printf("負の数です¥n");
```

```
a = 2*a;
```

単文はセミコロンで終わる
(単文以外はセミコロン不要)

## 複文

波カッコで複数の単文をひとまとめ

```
{
    b = 10;
    a = 2*b;
    printf("aは%dである¥n",a);
}
```

複文
単文
単文
単文

```
{
    a = 2*a;
}
```

a = 2*a; と単文で書いても同じ

if～else～文の文法

```
if (条件)
   文1
else
   文2
```

文1と文2はどちらも，単文でも複文でもOK

# if～else～文の書き方

```
if (条件)
   文1
else
   文2
```

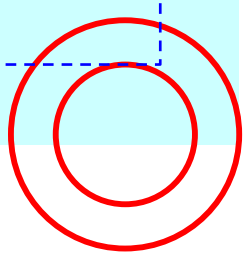
## 単文で書いた場合

```
#include <stdio.h>
int main(void)
{
  int data;

  scanf("%d", &data);
  if ( data < 0 )
      printf("負の数です¥n");   文1
  else
      printf("正の数です¥n");   文2
}
```

## 複文で書いた場合

```
#include <stdio.h>
int main(void)
{
  int data;

  scanf("%d", &data);
  if ( data < 0 )
  {                              文1
      printf("負の数です¥n");
  }
  else
  {                              文2
      printf("正の数です¥n");
  }
}
```

この例では単文で書いても複文で書いても意味の違いはないが，ミスが減るのでこちらがおススメ



# 複雑なif～else～文

```
if (条件) 文
   文1
else
   文2
```

```
if (条件1)
    文1
else
    if (条件2) 文2
        文2
    else
        文3
```

if文も全体で一つの文である.

```
int main(void)
{
    int x;
    printf("整数値xを入力してください：");
    scanf("%d", &x);
    if (x < 5)
    {
        printf("xの値は5未満です¥n");
    }
    else if (x >= 5 && x < 10)
    {
        printf("xの値は5以上10未満です¥n");
    }
    else
    {
        printf("xの値は10以上です¥n");
    }
}
```

文1　文2　文2　文3

整数値xを入力してください：3
xの値は5未満です

整数値xを入力してください：8
xの値は5以上10未満です

整数値xを入力してください：12
xの値は10以上です