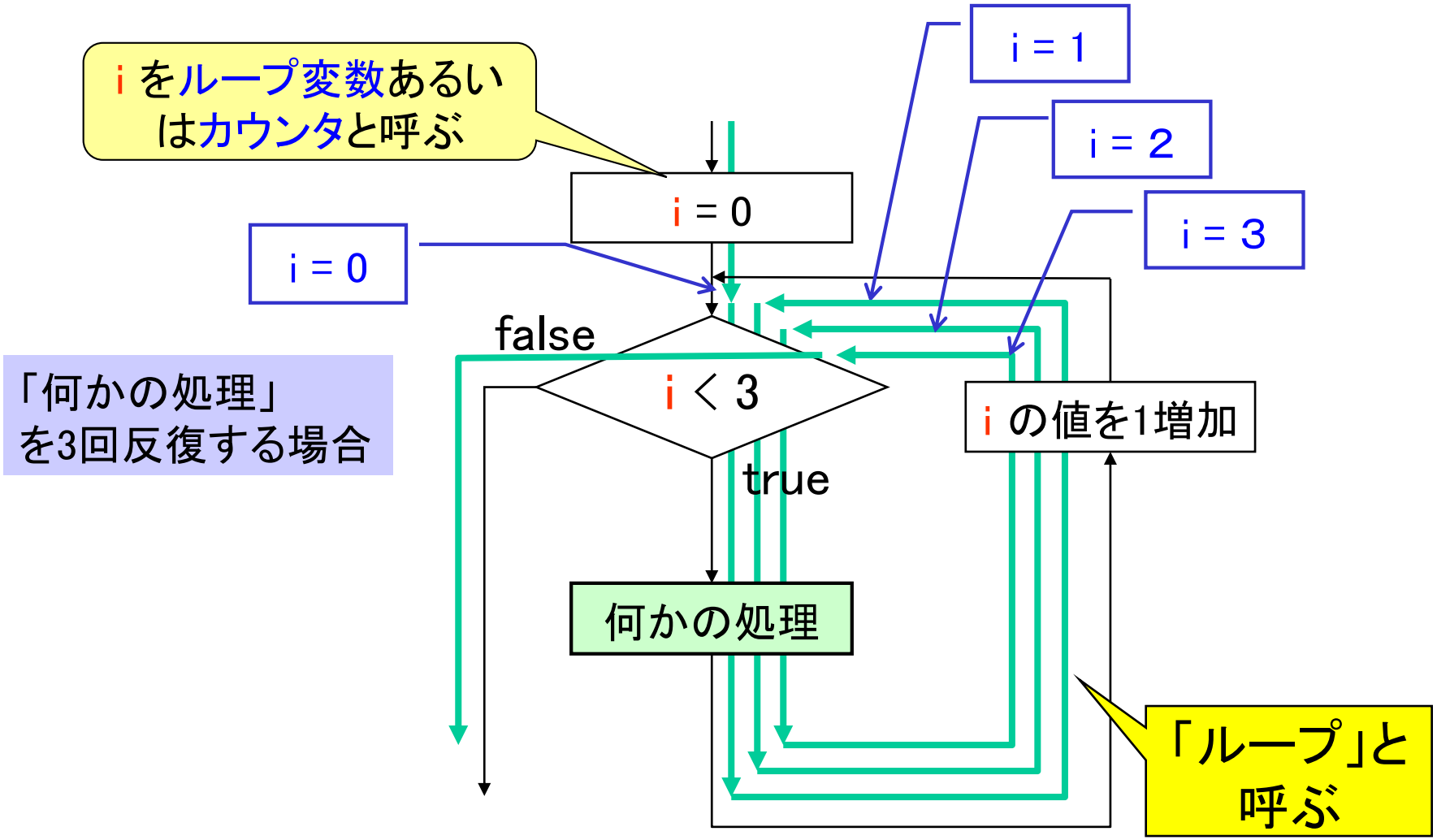
復習

一定回数を繰り返す反復処理の考え方



for文のパターン（ある回数だけ文を実行する場合）

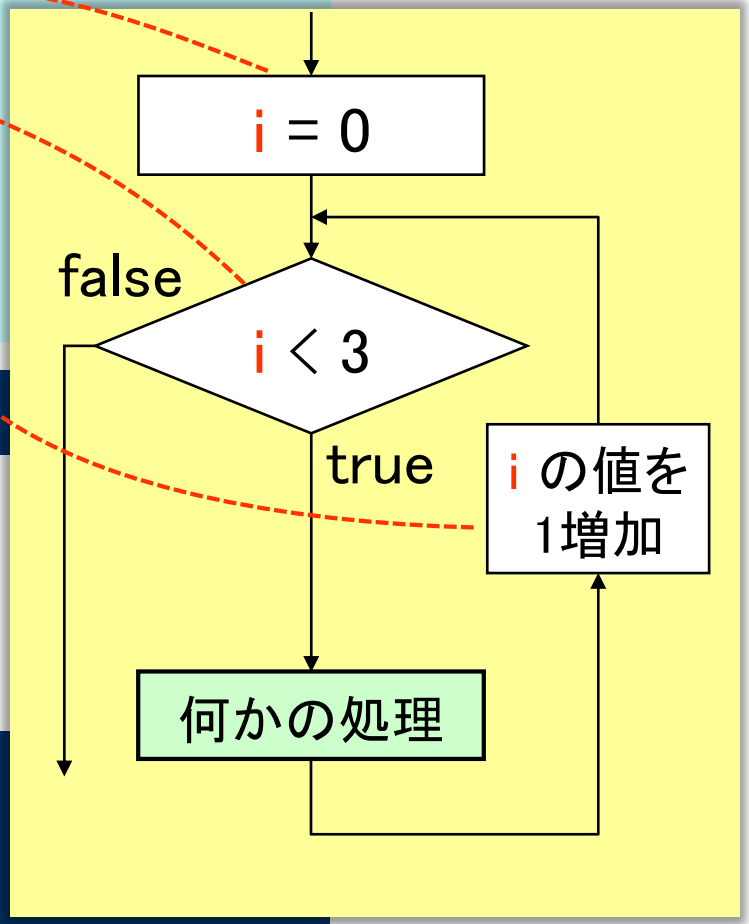
復習

for文のセミコロンは3パートの区切り
（単文の意味ではない）

キーワード　セミコロン　カッコが必要

```
int i;
for (i = 0; i < 3; i++)
{
    printf("A");
}
```

この中が3回実行される



AAA

ループ変数 i = 0 でスタートして
i < 3 の間は反復 ⇒ i = 0, 1, 2

何かの処理は3回実行される

```
int a;
for (a = 0; a < 5; a++)
{
    printf("B¥n");
}
```

```
B
B
B
B
B
```

ソース中の実行順のイメージ

復習

```
int i;
for ( i = 0 ; i < 3 ; i++ )
{
    printf("A");
}
```

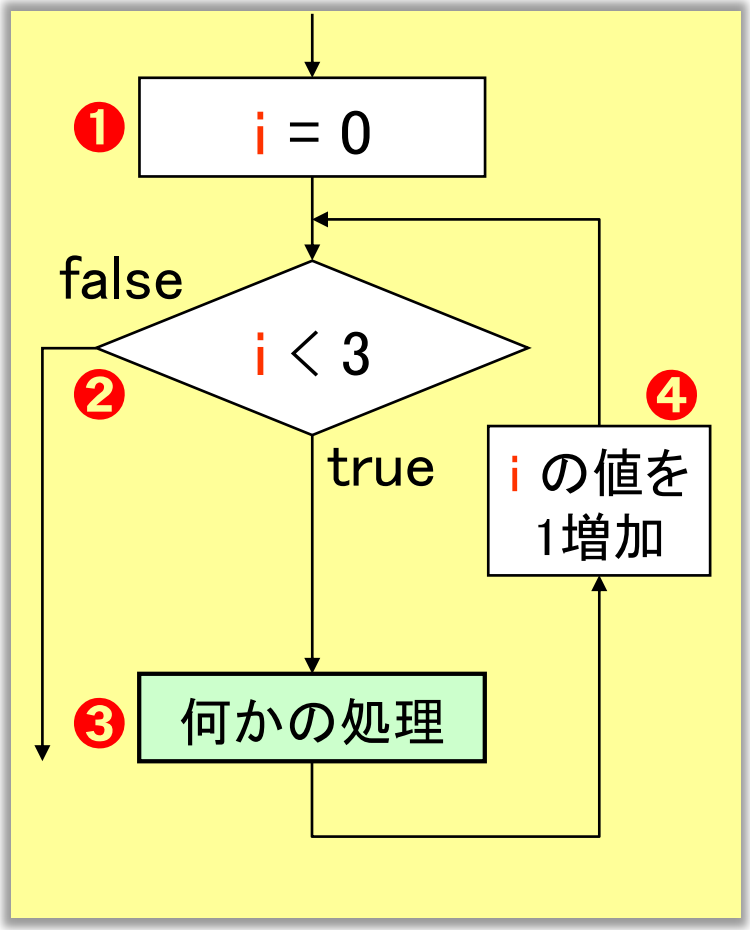
❶ ❷ ❸ ❹

False

True

AAA

！注意！
❷反復条件の設定によっては
1度もループしないことも！



復習

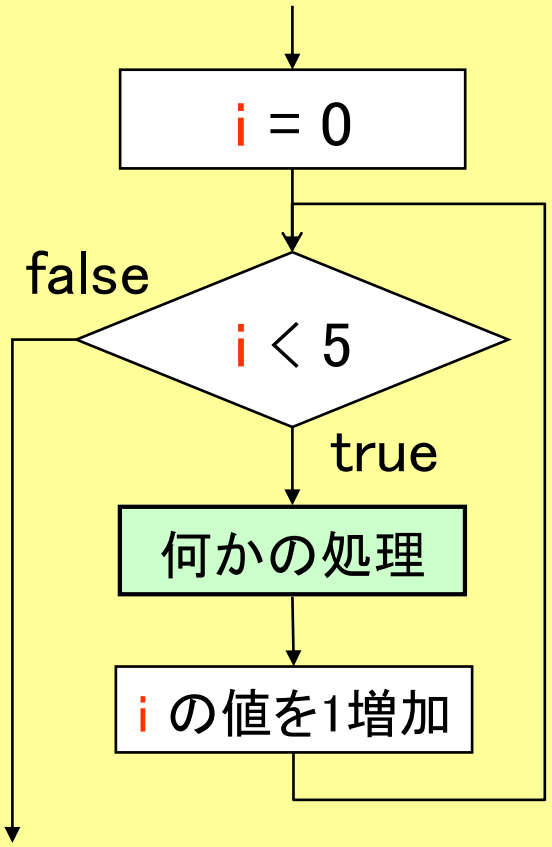
ループ変数(1)

```
int i;
for (i = 0; i < 5; i++)
{
    printf("%d¥n", i);
}
```

ループ変数iの値を表示

```
0
1
2
3
4
```

ループ変数の値は1づつ増加




```
int i;
for (i = 2; i < 5; i++)
{
    printf("%d¥n", i);
}
```

```
2
3
4
```

```
int i;
for (i = 3; i < 7 ; i++)
{
    printf("%d¥n", i);
}
```

```
3
4
5
6
```

初期値

繰り返しの条件
(反復条件)

復習

ループ変数(3)

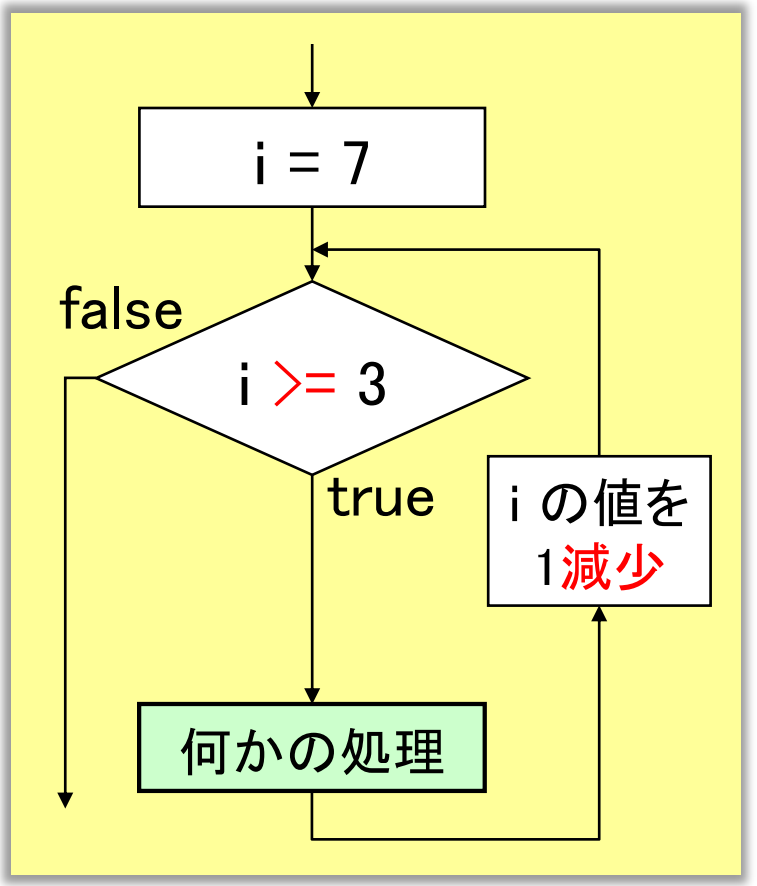
ループ変数の値が減っていく反復はどう書く？

```
int i;
for (i = 7; i >= 3; i--)
{
    printf("%d¥n", i);
}
```

```
7
6
5
4
3
```

ループ変数の
初期値は7

ループ変数が3より大きいか
等しい間は繰り返す！



a++ → 変数aの値を1増やす
a-- → 変数aの値を1減らす

復習

ループ変数(4)

i=i+1は変数iの値を1増やす．i++と同じ．

```
int i;
for (i = 3; i <= 7; i=i+1)
{
    printf("%d¥n", i);
}
```

```
3
4
5
6
7
```

ちょっと応用問題
次のプログラムの実行結果はどうなる？

```
int i;
for (i = 3; i <= 10; i=i+2)
{
    printf("%d¥n", i);
}
```

```
3
5
7
9
```

ループ変数のステップ値は2

i=i+2は変数iの値を2増やす

for文の文法

for文の文法のまとめ

これ自体も一つの文！

```
for （初期値設定 ； 反復条件 ； ステップ値設定）
   文
```

復習

文 → 単文 or 複文

単文

```
printf("負の数です¥n");
a = 2*a;
```

複文

複数の文を
ひとまとめ

```
{
  b = 10;          単文
  a = 2*b;         単文
  if (a > 5)
  {                                  if文
     printf("aは%dである¥n",a);
  }
}
```

if文も一つの文

問題:このプログラムは誤り？

```
int i;
for (i = 3; i <= 7; i=i+1)
    printf("%d¥n", i);
```

ループで実行する文が
一つならこれでもOK

よくある間違い

問題:この実行結果は？

```
int i;
for (i = 3; i <= 7; i++);
    printf("%d¥n", i);
```

こうなるはずでは？

```
3
4
5
6
7
続行するには . . .
```

違います！

実際にはこうなる！

```
8
続行するには . . .
```

この空文をi=3～7まで繰り返し実行！

わかりやすくソースを書き直すと

```
int i;
for (i = 3; i <= 7; i++)
;
printf("%d¥n", i);
```

i=8の時，ループを終了してこの文を一度だけ実行！

単文

```
printf("負の数です¥n");
a = 2*a;
;
```

セミコロンだけでも一つの文！（空文）

ソースプログラムにおける段付け

```
int main(void)
{
int i, x;
printf("整数値xを入力してください：");
scanf("%d", &x);
for (i = 0; i < x; i++)
{
printf ("i = %d¥n",i );
if (i < 5)
{
printf("5未満です¥n");
printf("まだ小さいです¥n");
}
else
{
printf("5以上です¥n");
printf("大きくなりました¥n");
}
}
}
```

段付け

```
int main(void)
{
    int i, x;
    printf("整数値xを入力してください：");
    scanf("%d", &x);
    for (i = 0; i < x; i++)
    {
        printf ("i = %d¥n",i );
        if (i < 5)
        {
            printf("5未満です¥n");
            printf("まだ小さいです¥n");
        }
        else
        {
            printf("5以上です¥n");
            printf("大きくなりました¥n");
        }
    }
}
```

カッコの位置は好みの問題

```
int i;
for (i = 3; i <= 10; i=i+2) {
    printf("%d¥n", i);
}
```

Tab キーの活用

段付けしたいところで
Tabキーを押す ⇒ 4文字分の段がつく

注)これはVisual Studioの標準設定の場合

ループ変数の利用(1)

1から5の整数値の和を求める　１＋２＋３＋４＋５＝１５

```
int a,sum = 0;
for (a = 1; a <= 5; a++)
{
    sum = sum + a;
    printf("a=%d  sum=%d¥n", a, sum);
}
printf("和は%d¥n", sum);
```

この例では，ループを始める前にsumが0であることが重要

大原則
初期化(代入)されていない変数の値は不定である.

- ゼロとは限らない
- どんな値が入っているか決まっていない
- 実行するたびに異なった値になることも

代入前のsumの値

aの値

足し算の結果

```
a=1   sum=1
a=2   sum=3
a=3   sum=6
a=4   sum=10
a=5   sum=15
和は15
```

```
sum ← 0                  + 1
sum ← 1 (=0+1)           + 2
sum ← 3 (=0+1+2)         + 3
sum ← 6 (=0+1+2+3)       + 4
sum ← 10 (=0+1+2+3+4)    + 5
```

sum=(0+1+2+3+4)+5

ループ変数の利用(2)

2から10の偶数の和を求める

```
int a,sum = 0;
for (a = 2; a <= 10; a = a+2)
{
    sum = sum + a;
    printf("a=%d  sum=%d¥n", a, sum);
}
printf("和は%d¥n", sum);
```

代入前のsumの値

aの値

```
a=2   sum=2
a=4   sum=6
a=6   sum=12
a=8   sum=20
a=10  sum=30
和は30
```

```
sum ← 0          +  2
sum ← 2 (=0+2)   +  4
sum ← 6 (=0+2+4) +  6
sum ←12 (=0+2+4+6)  +  8
sum ←20 (=0+2+4+6+8)  + 10
```

ループ変数を計算に利用しない計算

入力された5個の整数の和を求める

```
int  i, x, sum = 0;
for (i = 0; i < 5; i++)
{
    printf("値は？"); scanf("%d", &x);
    sum = sum + x;
}
printf("合計は%dです¥n",
```

代入前のsumの値

xの値

```
値は？ 3
値は？ 2
値は？ 7
値は？ 1
値は？ 12
合計は25です
```

```
sum ← 0                    +  3
sum ← 3 (=0+3)             +  2
sum ← 5 (=0+3+2)           +  7
sum ← 12 (=0+3+2+7)        +  1
sum ← 13 (=0+3+2+7+1)      + 12
```

反復処理(反復制御文)

プログラムにおける反復処理→
(a) ある処理を決まった回数繰り返す処理
(b) 一定の条件を満たす間繰り返す処理

繰り返す回数は決まっていない

ループ変数を利用するループ

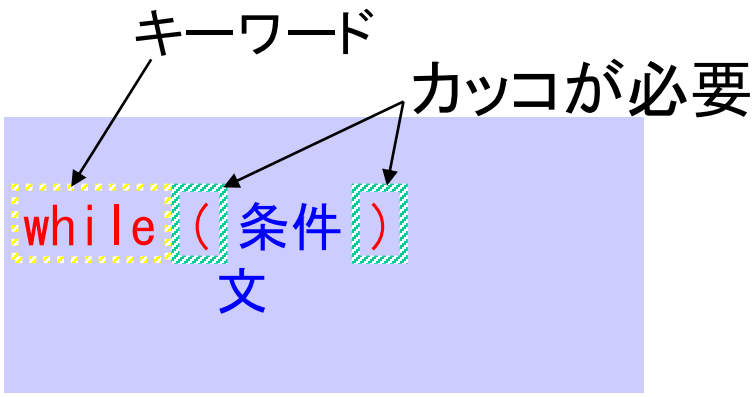
ループ変数を利用しないループ

C言語における反復処理 → 3種類

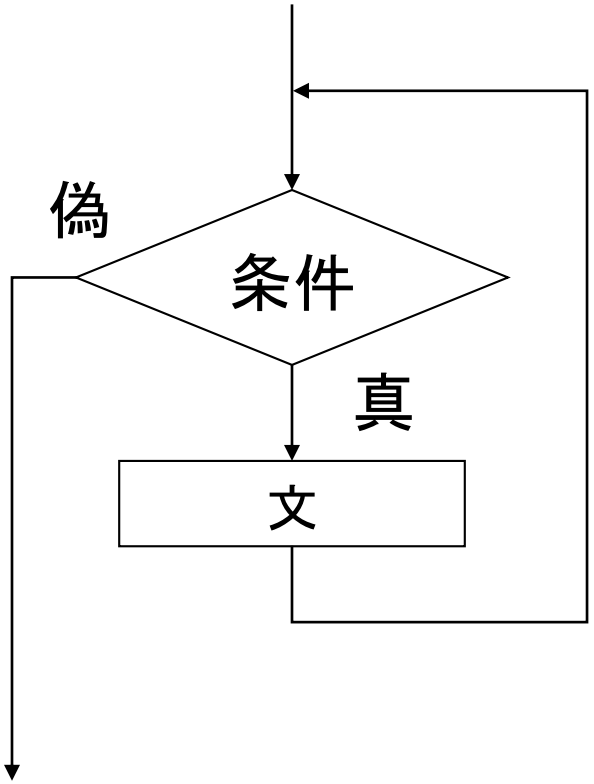
(a) 決まった回数反復処理を行う　for 文

(b) 条件が成り立つ間反復処理を行う
i) 反復処理の初めに条件を調べる場合　while 文
ii) 反復処理の終わりに条件を調べる場合　do while 文

while文(1)



・ 条件が真の間ずっと文(複文も可)を反復する
・ ループ変数は必要ない



while文(2)

ユーザーが0を入力するまで繰り返す

【注意】初めからaが0だと一度も実行されない

```
int a = 1;
while ( a != 0 )
{
    printf("正の整数を入力してください: ");
    scanf("%d", &a);
}
printf("終了します");
```

条件

```
int a = 0;
```

➡ 一度も実行されない

```
int a;
```

➡ 実行するたびに異なった結果！

条件が真(true)ならこの中が繰り返し実行される

```
正の整数を入力してください:5
正の整数を入力してください:6
正の整数を入力してください:10
正の整数を入力してください:15
正の整数を入力してください:0
終了します
```

無限ループ(1)

```
int i, j = 1;
while ( j != 0)
{
    printf("i = ? ");
    scanf("%d", &i);
}
```

```
i = ? 10
i = ? 5
i = ? 6
i = ? 15
i = ? 0
i = ? 0
i = ? 0
```

ループが終了しない！

常に真!

プログラムミスによる無限ループの例

```
while ( 3 >= 2 )
{
    printf("A¥n");
}
```

```
while (1)
{
    printf("A¥n");
}
```

C言語では0以外の定数値は真を意味する

プログラムミスによる無限ループからの強制脱出
→ Ctrl-C（Ctrlキーを押しながらCキーを押す）

デバッガの利用(1)

プログラムのミス
→ バグ

虫のこと

バグを見つけて修正すること
→ デバッグ

バグを発見する道具
→ デバッガ

どんな優秀なプログラマーでも、初めからバグのない完全なプログラムは作れない！
⇒ プログラムを完成させることはデバッグを完了すること！

プログラミング ≠ プログラムを書く

プログラミング ≒ デバッグ

自分で考え抜いてデバッグすることが重要

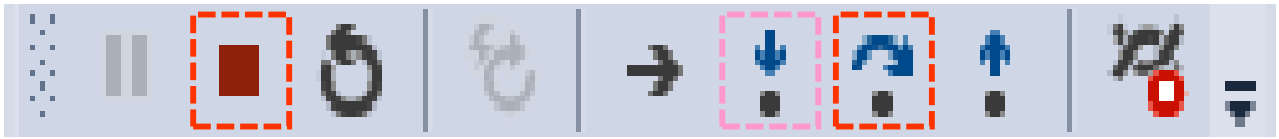
デバッガの利用(2)

大原則
バグがあったら，デバッガを起動すること！

[A] ツールバーでデバッグを始める手順

(1) ツールボタンで右クリックして，デバッグを選ぶ

(2) デバッグツールバーからボタンを選ぶ



(3)　　(2)　(1)

(1)ステップオーバー　　次の行を実行
(2) ステップイン　次の行を実行(今回は使わない)
(3) デバッグの中止　　デバッグをやめる

[B] メニューでデバッグを始める手順

デバッグメニューで，ステップオーバーを選ぶ