シラバス

5,6. 繰り返し処理の実現方法及び復習 繰り返し処理の実現方法及び複合代入演算子の確認
事前学修：授業スライド「第3回　繰り返し処理の実現方法及び復習」と教科書p.61～p.72の熟読。（60分）
事後学修：授業スライドの用語とその意味の理解。反復処理のフロー，while文，for文，do-while文によるプログラミング法，複合演算子を用いた条件の書き方を説明できること。（60分） 演習課題のプログラム及びレポートの完成と提出。（180分）

プログラミングの基礎及び演習

情報工学科

プログラミングの基礎 第3回

- 反復処理
  - 反復処理の復習
  - 反復処理をC言語で表現するには？
    - while文
    - do-while文
  - もう一つの反復処理: for文

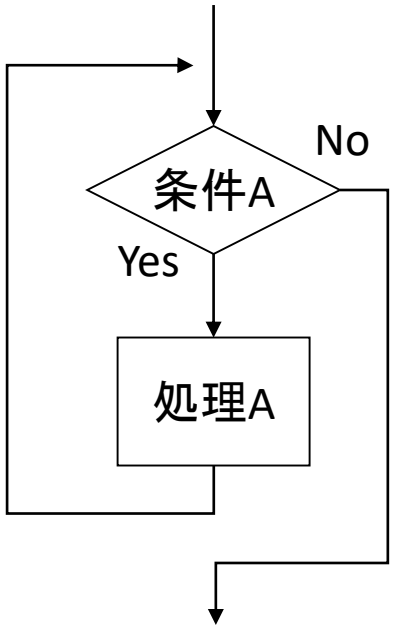
- 教科書p.61～p.72参照

反復処理とは？

- ある条件が真である間，処理を繰り返す
  - 条件と処理の手順の例:
    1. 三角形の底辺の長さを入力する
    2. 処理1で入力した値が負またはゼロの場合，
       処理1に戻る
    3. 三角形の高さを入力する
    4. 処理3で入力した値が負またはゼロの場合，
       処理3に戻る
    5. 処理1と3で入力した値を用いて，
       三角形の面積を計算する

反復処理: 前判定型反復処理

- 前判定型反復処理:
  繰り返し処理の先頭で条件判定



条件Aが真であるか調べる

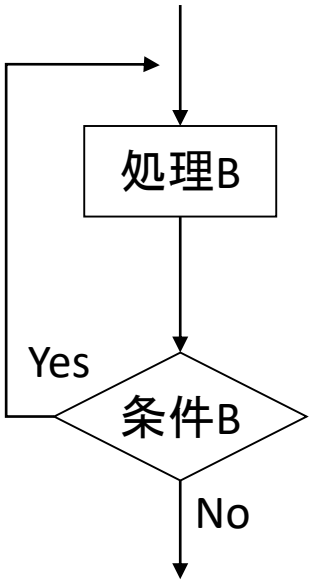
真である
→　処理Aを実行する
→　もう一度条件判定

偽である
→　処理Aを実行しない
→　次へ行く

処理Aには，複数の処理を入れることも可能

反復処理: 後判定型反復処理

- 後判定型反復処理:
  繰り返し処理の最後で条件判定



処理Bを実行する

次に，条件Bが真であるか調べる

真である
→　処理Bに戻る

偽である
→　繰り返しを終了し，次へ

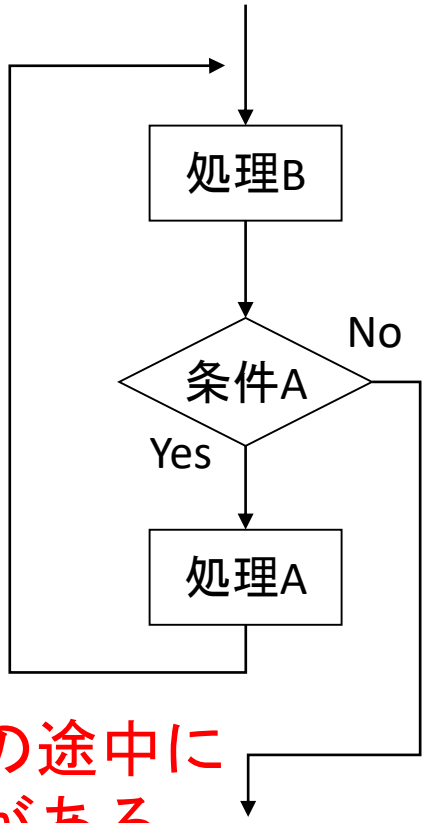
処理Bには，複数の処理を入れることも可能

反復処理のポイント

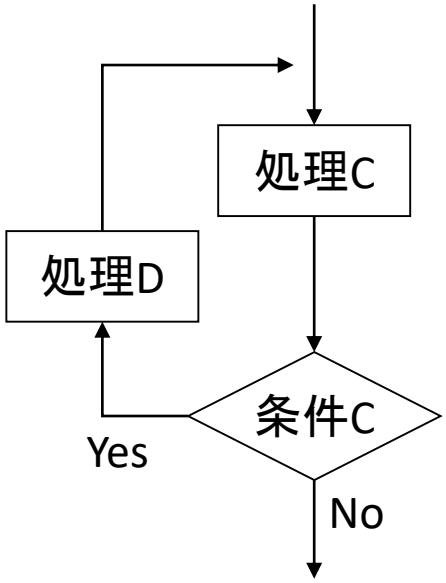
- 前判定型反復処理
  - 繰り返し処理の先頭で条件判定
  - 1回も処理を実行しない場合もある
- 後判定型反復処理
  - 繰り返し処理の最後で条件判定
  - 最低1回は必ず処理を実行する

可能な限り，前判定型反復処理にする
また上記以外の形にするのは，できるだけ避ける

反復処理： 例外



繰り返しの途中に
条件判定がある

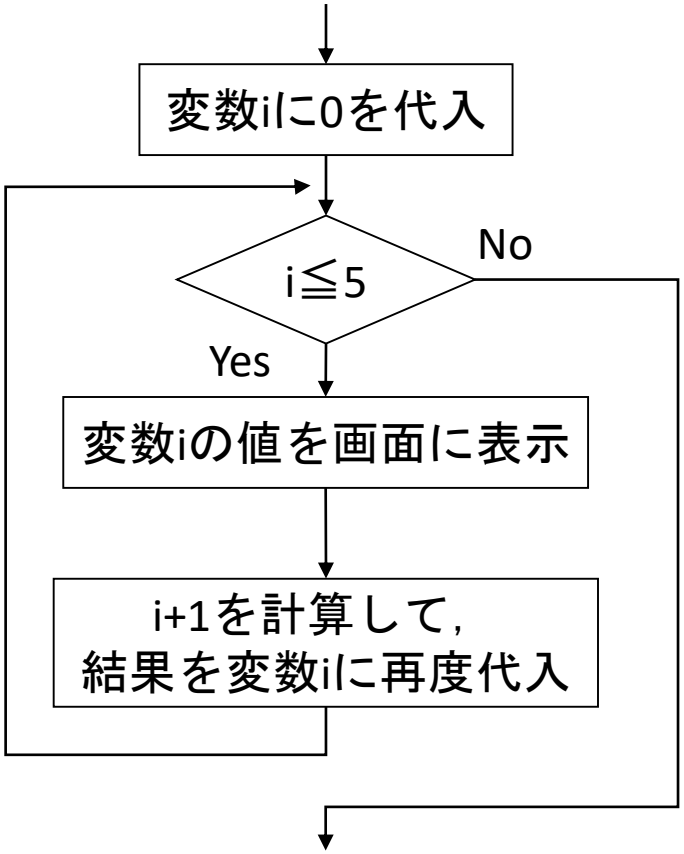


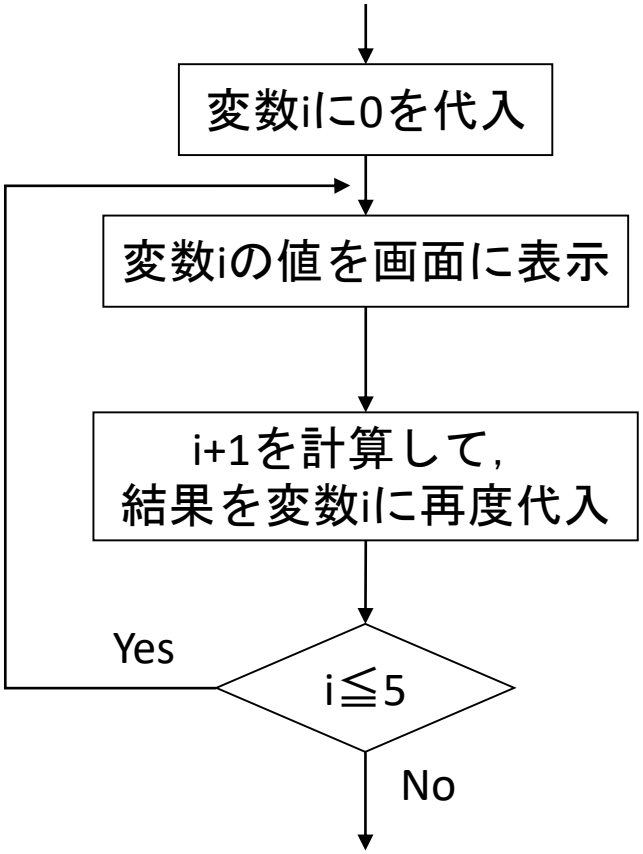
先頭に戻る途中に処理がある
→　書き方が違うだけで，左図と同じ

この形式にするのは，やむを得ない場合のみ

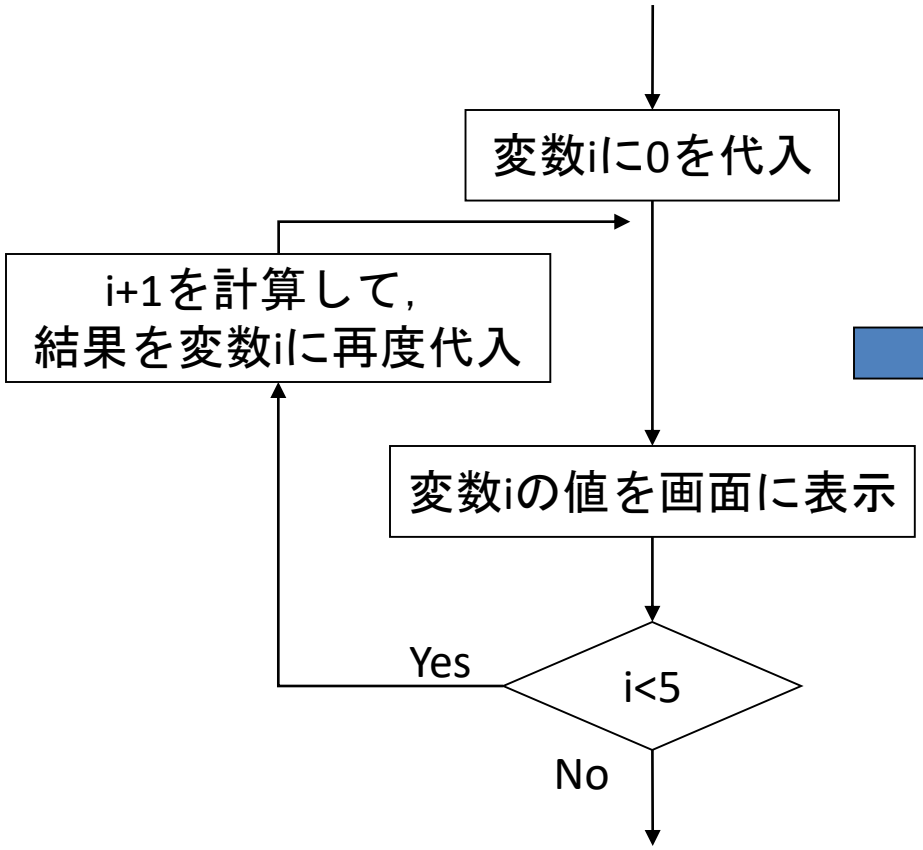
フローチャートの例

0から5まで順に数字を数えるプログラム

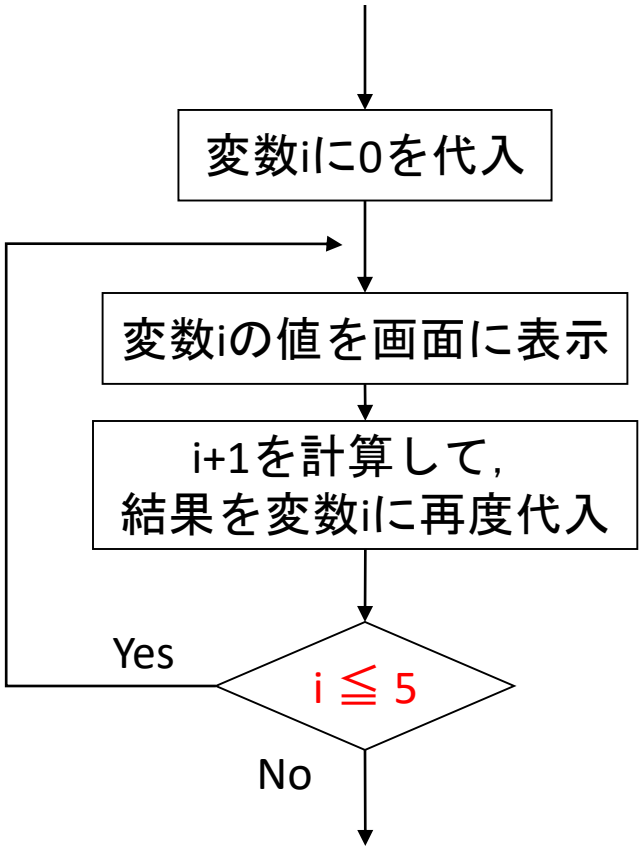




フローチャート：　良くない例



条件判定が繰り返しの途中にある



これならOK

プログラミングの基礎 第3回

- 反復処理
  - 反復処理の復習
  - 反復処理をC言語で表現するには？
    - while文
    - do-while文
  - もう一つの反復処理: for文

- 教科書p.61～p.72参照

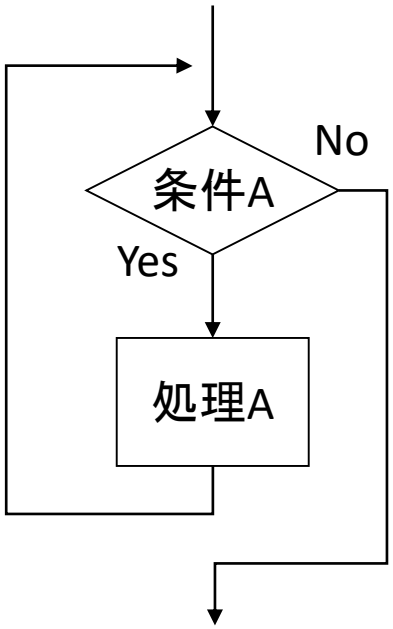
C言語での前判定型反復処理

- 前判定型の反復処理はwhile文で表現する
  - 書式:

```
while (条件A)
    処理A;
```

  - 条件の書き方はif文と同じ
  - ブロックも使える

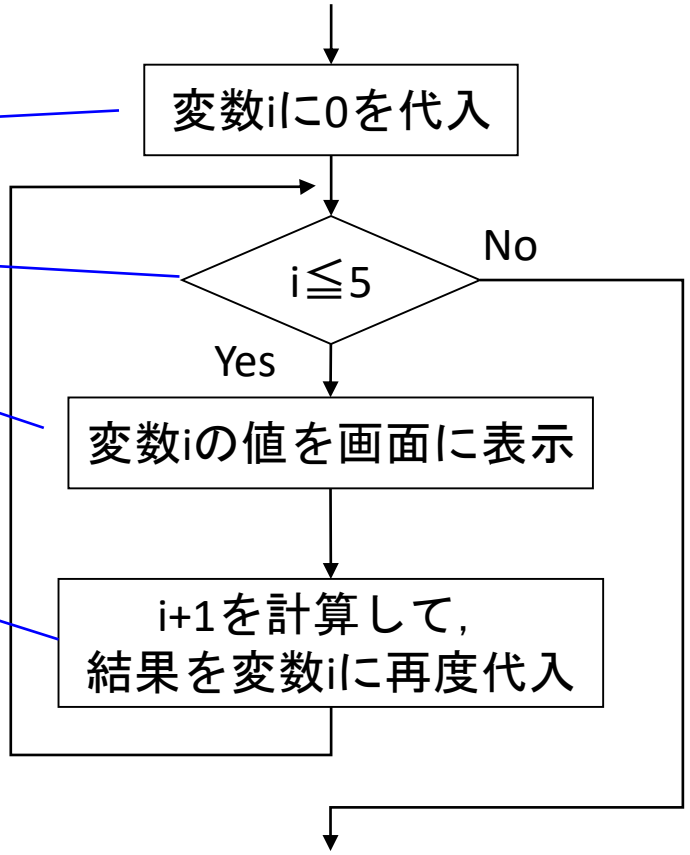
```
while (条件A) {
    処理A;
    処理B;
}
```



while文の例 (1)

例題5.1

```
i = 0;
while ( i <= 5 ) {
    printf( "i=%d¥n", i );
    i = i + 1;
}
```



while文の例 (2)

例題5.1

```
i = 0;
while ( i <= 5 ) {
    printf( "i=%d¥n", i );
    i = i + 1;
}
```

i = 0

i <= 5 ⇒ 真なので繰り返し
printf ⇒ "i=0" を表示
i = i + 1 ⇒ i = 1 になる

i <= 5 ⇒ 真
printf ⇒ "i=1" を表示
i = i + 1 ⇒ i = 2 になる

i <= 5 ⇒ 真
printf ⇒ "i=2"を表示
i = i + 1 ⇒ i = 3 になる

中略

i = 6 になる
i <= 5 ⇒ 偽なので終了

複合演算子 (1)

- i=i+1 という形はC言語では多用される
  - 簡略記法が用意されている
    - i=i+1はi+=1と書いても良い．i=i+1でも構わない．

演算子	例	意味
+=	x += y	x = x + y
-=	a -= b	a = a - b
*=	v *= w	v = v * w
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2

複合演算子 (2)

- `i+=1` や `i-=1` は，
  複合演算子の中でも特によく使う
  - さらに簡略化した記法が可能

通常の記法	`i = i + 1`	`i = i - 1`
簡略記法	`i += 1`	`i -= 1`
	`i++`	`i--`
	`++i`	`--i`

複合演算子(3)

- i++と++i
  - どちらもi=i+1の計算を行う
  - 教科書の表5.3（p.64）の違いがある
    - どのタイミングで加算処理を行なうか
  - この違いを利用したプログラムを書くことは推奨しない
    - 分かりにくくなるため
    - プログラムで重要なのは，「分かりやすさ」
      - 記述が多少長くなっても，分かりやすく書く方が重要

プログラミングの基礎 第3回

- 反復処理
  - 反復処理の復習
  - 反復処理をC言語で表現するには？
    - while文
    - do-while文
  - もう一つの反復処理: for文

- 教科書p.61～p.72参照

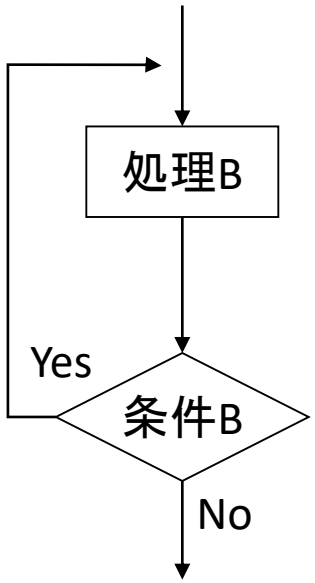
C言語での後判定反復処理

- 後判定型の反復処理はdo-while文で表現する
  - 書式

```
do {
    処理B;
 } while( 条件B );
```

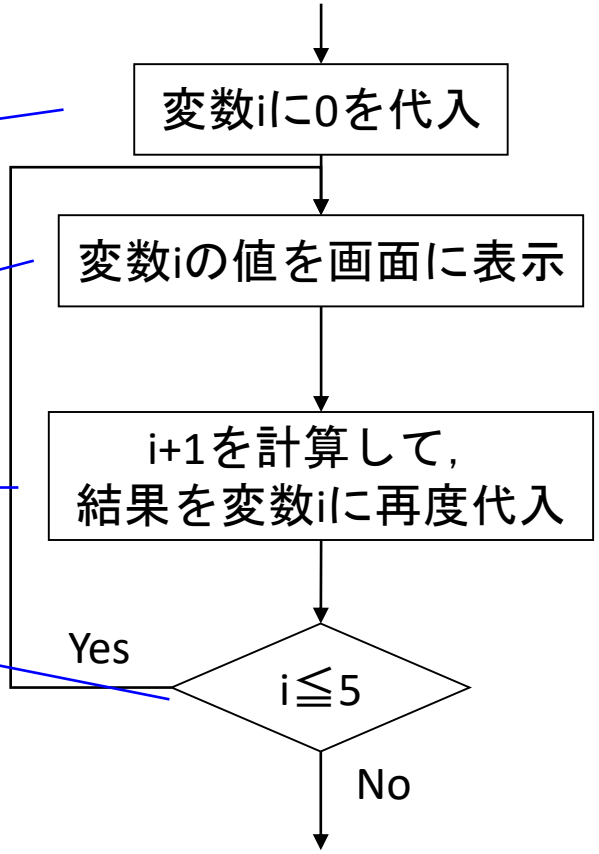
  - 通常，ブロックを用いる

```
do {
    処理B;
    処理C;
    …
 } while( 条件B );
```



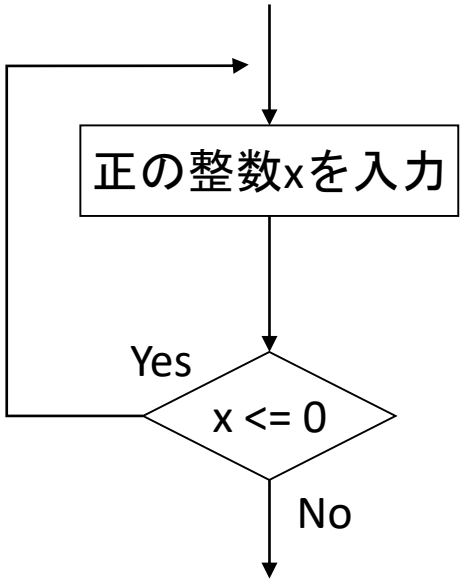
do-while文の例 (1)

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



do-while文の例 (2)

- データ入力時に，
  入力条件に合致するかどうか
  チェックするときに使える
- 例：
  - 正の整数を入力させたい
  - 負・ゼロなら再入力




```
do {
    printf( "正の整数を入力: " );
    scanf( "%d", &x );
} while( x <= 0 );
```

無限ループ (1)

- 無限ループとは？
  - 永久に繰り返される反復処理
  - 強制終了させないと終わらない
    - UNIXなら[Ctrl-C]で強制終了
- while文で，常に”真”となる条件を記述すると無限ループになる
  - 例
    - `while (1 > 0)`
    - `while (1)`
      - C言語では，単に“1”と書けば真という意味になる

無限ループ (2)

- 無限ループからの脱出方法
  - break文を使う　⇒　ループからの強制脱出

例題5.2

```
int x;
x = 0;
while ( 1 ) {
    printf( "x = %d¥n", x );
    if( x == 3 )
        break;
    x++;
}
```

printf　⇒　"x = 0"　と表示
if(x == 3)　⇒　偽
x++　⇒　x = 1

printf　⇒　"x = 1"　と表示
if(x == 3)　⇒　偽
x++　⇒　x = 2

printf　⇒　"x = 2"　と表示
if(x == 3)　⇒　偽
x++　⇒　x = 3

printf　⇒　"x = 3"　と表示
if(x == 3)　⇒　真　breakにより終了

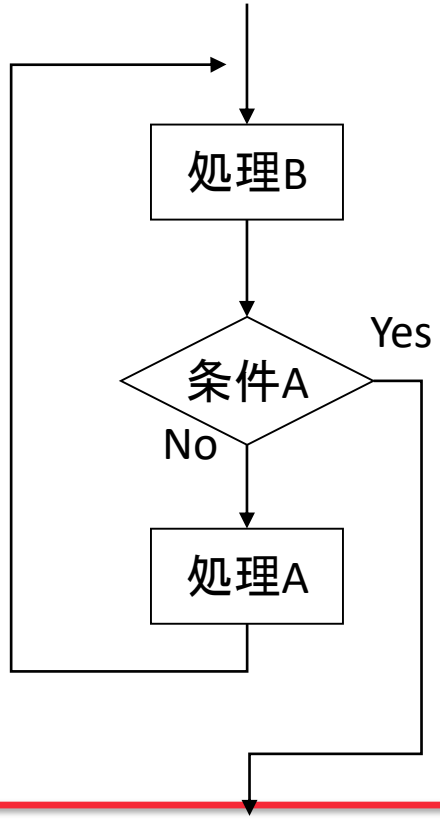
例外型の反復処理

- 繰り返しの途中に条件判定がある場合
  - breakを使うと，例外型の反復処理を作れる

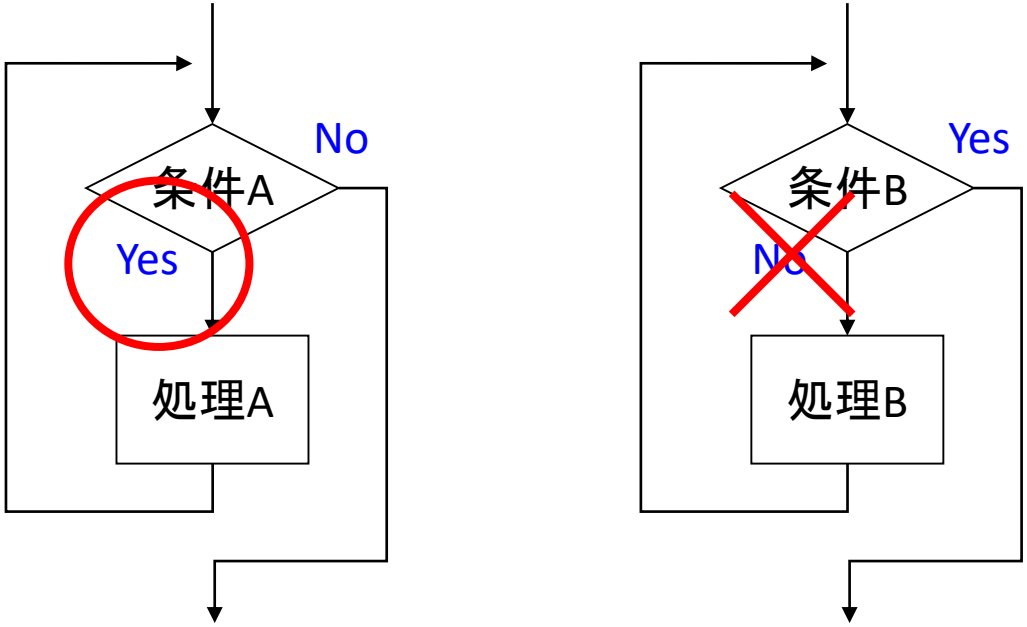
```
while ( 1 ) {
    処理B;

    if( 条件A )
        break;

    処理A;
}
```

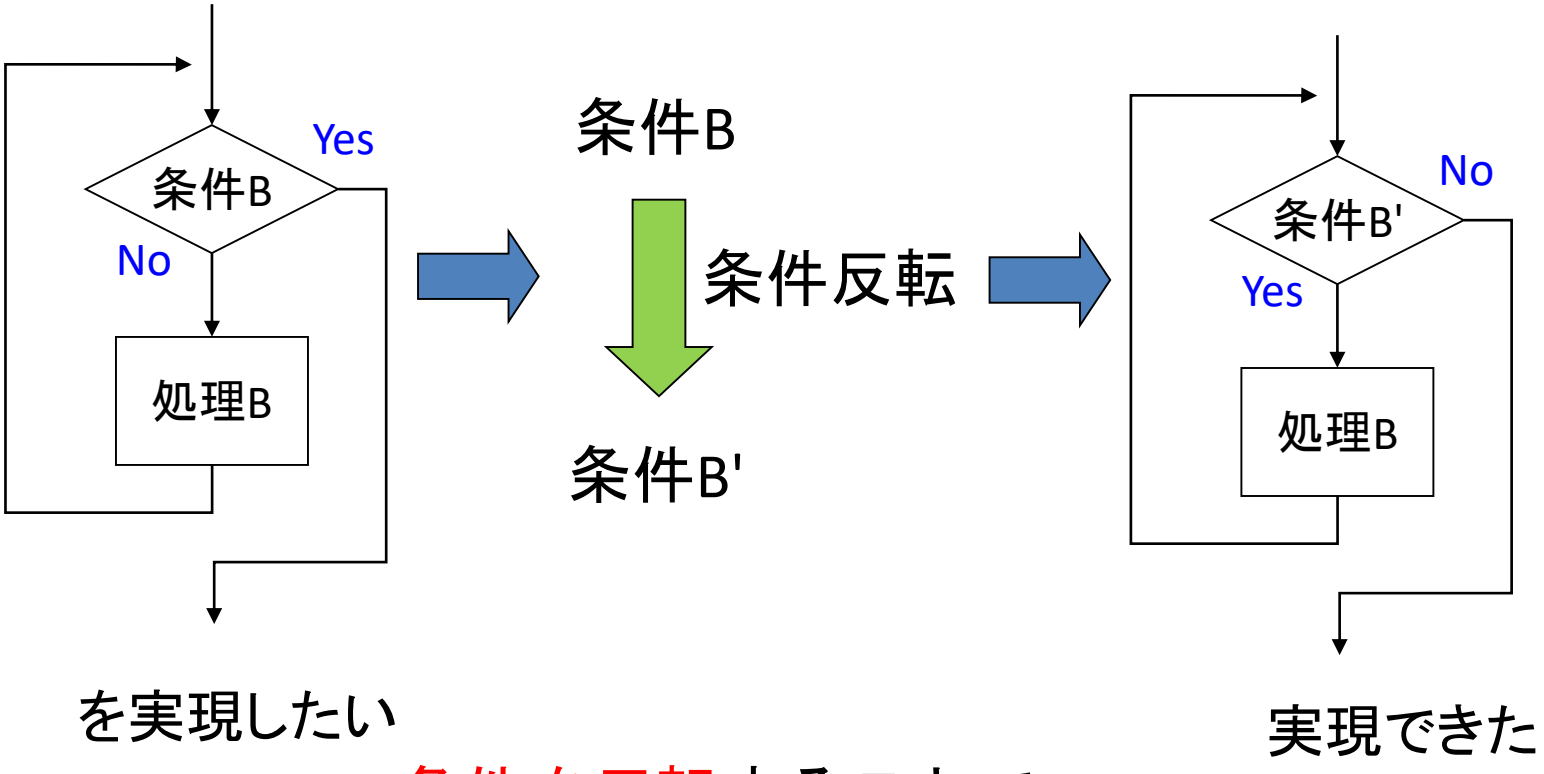


反復処理の条件 (1)



C言語の反復処理では，必ずYes側が繰り返し処理になる．
No側を繰り返しにすることはできない

反復処理の条件 (2)



を実現したい

実現できた

条件を反転することで，
Yes/Noを逆にする

条件の反転

反転前	反転後
x < y	x >= y
x >= y	x < y
x == y	x != y
(x >= y) && (a == b)	(x < y) || (a != b)
(x > y) || (a >= b)	(x <= y) && (a < b)

「ド・モルガンの法則」

- 等号つきの不等号は等号を外して逆向き
- 等号なしの不等号は等号をつけて逆向き
- 論理積は論理和にして，各要素を反転
- 論理和は論理積にして，各要素を反転

補足: 条件式の値

- 条件式の評価の結果
  - 真なら1
  - 偽なら0

- 例： xが10でyが5の場合の条件式の評価
  - x > y は 1
  - x < y は 0

C言語では，if文やwhile文の条件式の評価の結果が0（偽になる）か0以外（真になる）で判定している

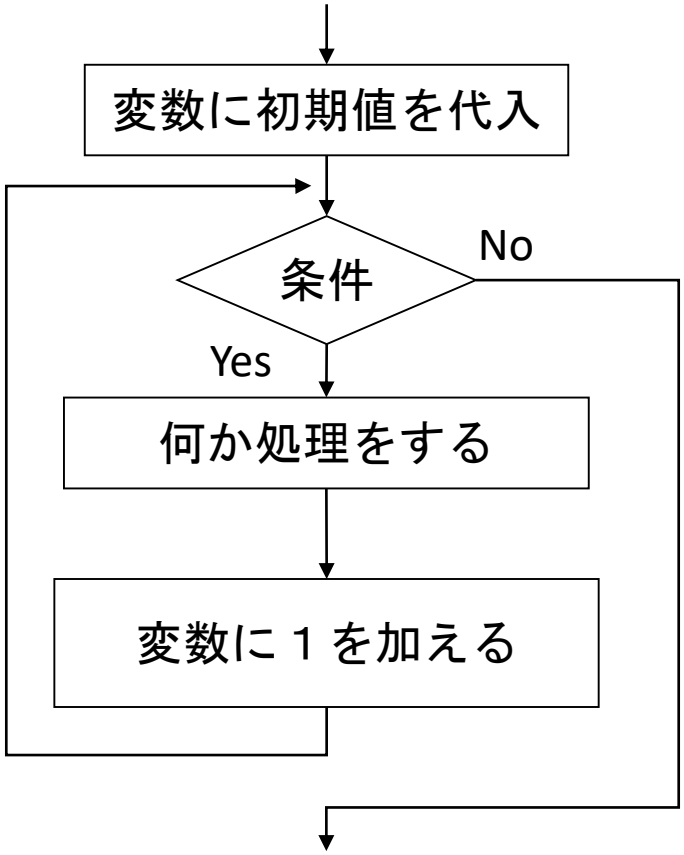
プログラミングの基礎 第3回

- 反復処理
  - 反復処理の復習
  - 反復処理をC言語で表現するには？
    - while文
    - do-while文
  - もう一つの反復処理: for文

- 教科書p.61～p.72参照

while文の定型パターン

```
変数に初期値を代入
while（ 条件 ）{
        何か処理をする
        変数に1を加える
}
```



「変数の初期化」，「条件判定」，「変数に1を加える」
処理を1つにまとめる → for文

for文

- 書式:

```
for（初期値の代入; 条件; 増分の処理)
   処理A;
```

  - 条件の書き方はif文と同じ
  - ブロックも使える

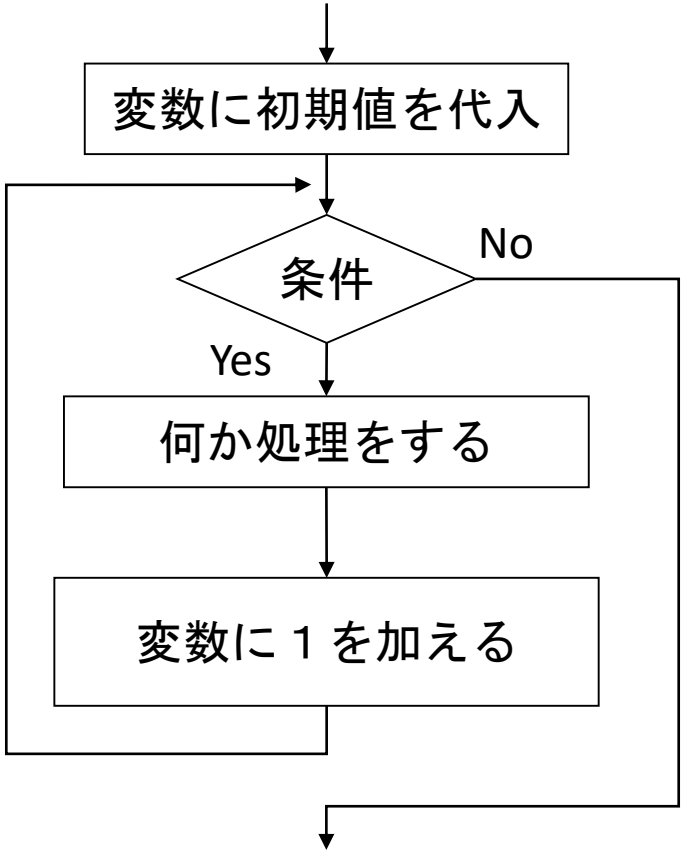
```
for（初期値の代入; 条件; 増分の処理)
{
   処理A:
   処理B;
}
```

for文

```
for ( 初期値の代入; 条件; 変数に１を加える)
    何か処理をする ;
```

同じ

```
初期値の代入;
while ( 条件 ) {
    何か処理をする ;
    変数に１を加える;
}
```

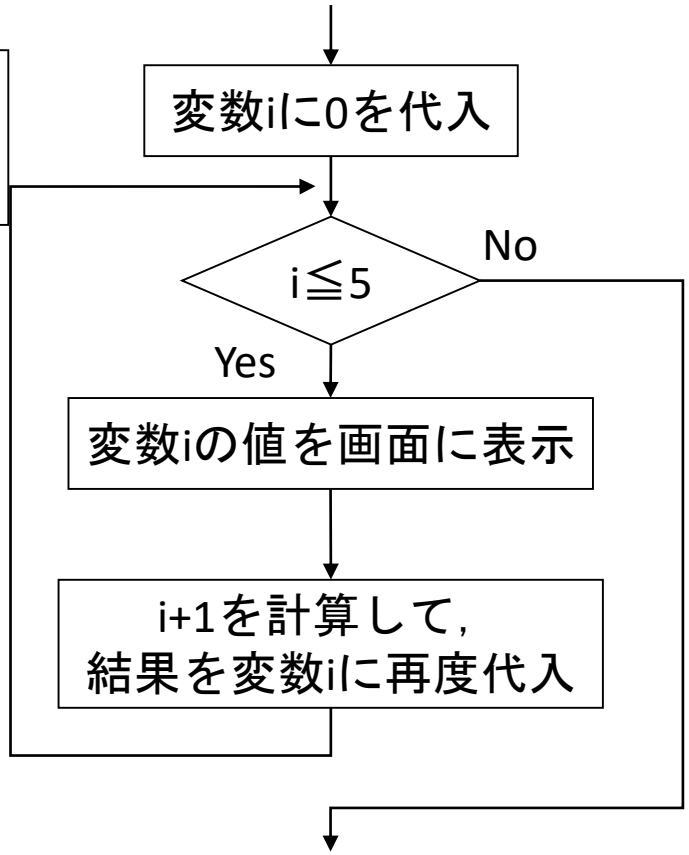


for文の例

- 0から5まで順に数字を数えるプログラム

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

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



例題：　合計の計算 (1)

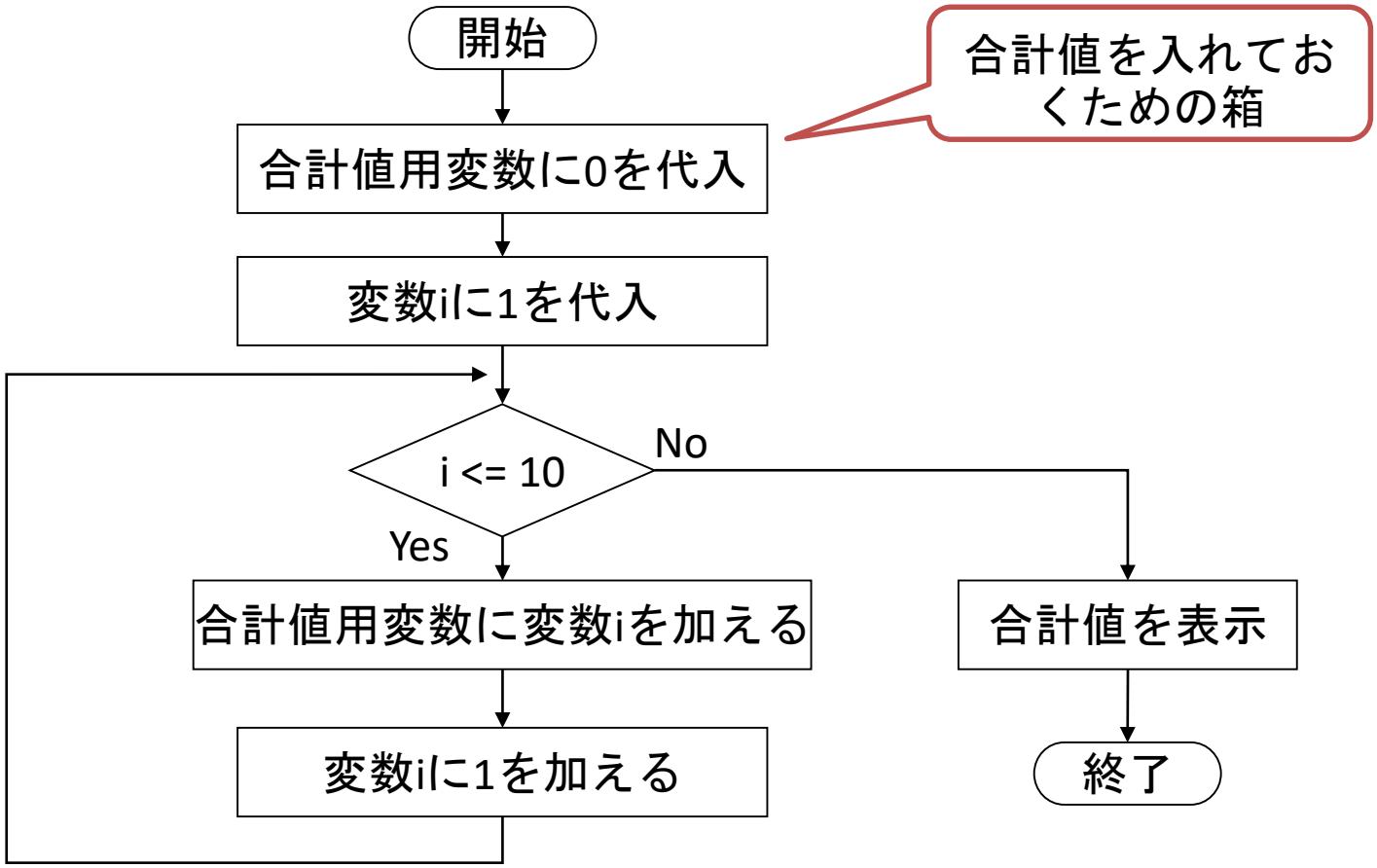
- 1から10までの整数の和を求めるには？

```
sum = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
```

これでもOK
だが，1から10,000までの合計なら？
1から100,000,000までの合計は？

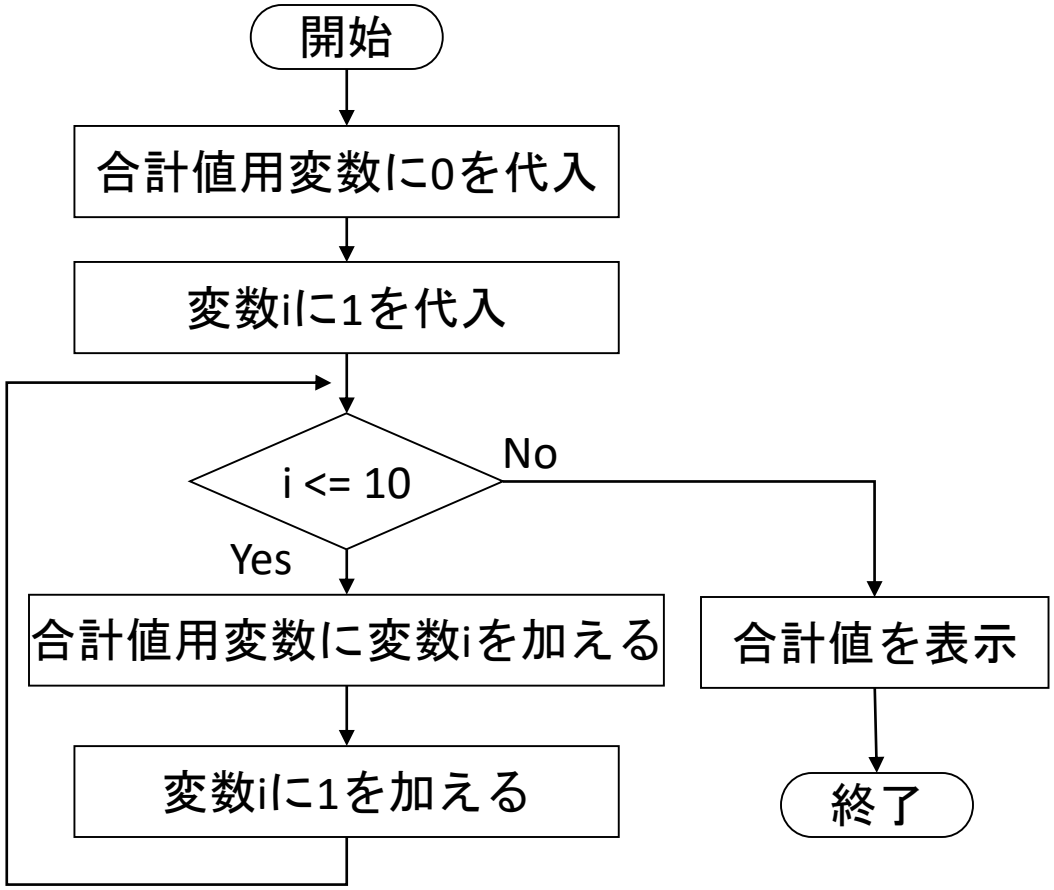
反復処理を使うと，より汎用的に記述できる

例題：　合計の計算 (2)


開始
合計値用変数に0を代入
変数iに1を代入
i <= 10
No
Yes
合計値用変数に変数iを加える
変数iに1を加える
合計値を表示
終了
合計値を入れておくための箱

例題：　合計の計算 (3)

合計 = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10



i	合計
	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

例題：　合計の計算 (4)

重要！

例題5.4

```
int i, sum;
sum = 0;
for ( i = 1; i <= 10; i++ )
    sum += i;
printf( "合計 = %d¥n", sum );
```

合計用変数に初期値0を代入

反復条件: i <= 10なら繰り返し

変数iに1を代入

合計用変数に
変数iを加える

合計値の表示

例題：　合計の計算 (5)

while文を使ってもよい

```
int i, sum;
sum = 0;
i = 1;
while ( i <= 10 ) {
    sum += i;
    i++;
}
printf( "合計=%d ¥n", sum );
```

forで書ける処理は，
whileでも書ける．

whileで書ける処理は，
forでも書ける．

どちらを使っても構わない
使いやすい方を使う．

例題：　合計の計算 (6)

- forやwhileを使って，
  合計を計算するメリット
  - 「1からNまで」のNを簡単に変更できる．

この数字を変えるだけで，
任意の数の合計が求められる

```
int i, sum;
sum = 0;
for ( i = 1; i <= 10; i++ )
    sum += i;
printf( "合計 = ¥%d¥n", sum );
```

例題：　平均の計算 (for版）　(1)

- 平均値　＝　合計　÷　データ数

→合計を求めてからデータ数で割れば良い

```
int i;
double x, sum=0.0, ave;
for ( i = 1; i <= 5; i++ ) {
    printf( "input %d = ", i );
    scanf( "%lf", &x );
    sum += x;
}
ave = sum / 5.0;
printf( "平均 = %f¥n", ave );
```

データを５つ入力

データの合計を求める

データ数で割る

例題：　平均の計算 (for版) (2)

- 割り算の注意事項：
  - 実数÷実数　→　実数
  - 整数÷整数　→　整数　（小数点以下切捨て）
  - 実数÷整数　→　実数
  - 整数÷実数　→　実数
  - 例：
    - `5.0 / 2.0` ⇒ `2.5`
    - `5 / 2` ⇒ `2`
    - `5.0 / 2` ⇒ `2.5`
    - `5 / 2.0` ⇒ `2.5`

整数: int型
実数: double型

キャスト

- 「整数÷整数」で、結果を実数にするには？
- キャストを使う
  - キャスト: 強制的に型を変えること
  - 書式

    （型の名前） 変更した変数

    例:

```
int x = 5, y = 2;
double z;
z = (double) x / y;
```

int型の変数xを
「強制的にdouble型に変更」.
右辺の計算結果はdouble型となる.

キャストによる平均値の計算 (for版)

```
int i;
double x, sum = 0.0, ave;
for ( i = 1; i <= 5; i++ ) {
    printf( "input %d = ", i );
    scanf( "%lf", &x );
    sum += x;
}
ave = sum / 5.0;
printf( "平均 = %f¥n", ave );
```

sumがdouble型なので，
ave = sum / 5; でもOK
（double/int はdoubleになる）

```
int i, sum = 0, x;
double ave;
for ( i = 1; i <= 5; i++ ) {
    printf( "input %d = ", i );
    scanf( "%d", &x );
    sum += x;
}
ave = (double) sum / 5;
printf( "平均 = %f¥n", ave );
```

sumがint型なので，
ave = sum / 5; ではダメ
（int/int はintになる）

キャストによる平均値の計算 (for版)

```
int i;
double x, sum = 0.0, ave;
for ( i = 1; i <= 5; i++ ) {
    printf( "input %d = ", i );
    scanf( "%lf", &x );
    sum += x;
}
ave = sum / 5.0;
printf( "平均 = %f¥n", ave );
```

```
int i, sum = 0, x;
double ave;
for ( i = 1; i <= 5; i++ ) {
    printf( "input %d = ", i );
    scanf( "%d", &x );
    sum += x;
}
ave = (double) sum / 5;
printf( "平均 = %f¥n", ave );
```

int型変数sumを
一時的にdouble型に
変更する

for文の補足: 変数に値を加える

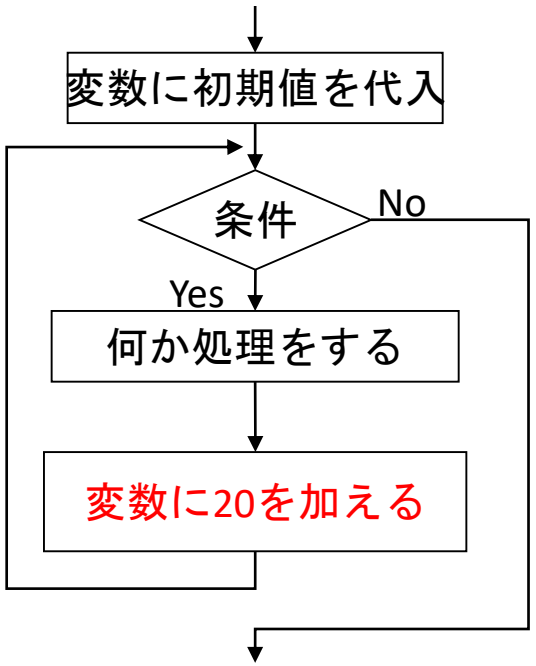
for (初期値の代入; 条件; 変数に１を加える)
　何か処理をする ;

「変数に１を加える」以外の繰り返しパターンもある

```
for ( i = 0; i <= 100; i += 20 ) {
    printf( "i = %d¥n", i );
}
```

```
i = 0
i = 20
i = 40
i = 60
i = 80
i = 100
```

実行結果



【演習課題】

- 演習課題提出システムの「第3回」演習課題を実施
- 演習課題3-1～3-6
  - 演習時間に，設計後に演習環境にリモートログインしプログラムを作成
  - 演習環境でコンパイル，テストし，完成
  - ソースファイル（3-x.c）をWinSCP等でPCへ転送
  - 課題提出システムへSubmitし各課題100点を目指す
  - 時間内に終了しない場合　⇒　宿題
- 授業翌週水曜日の午後5時までに必ず提出

【レポート】

- 演習課題提出システムの
  演習課題3-4,3-5について，下記内容を記載し，印刷し，授業時に配布した表紙をつけた，
  ”レポート”を提出せよ.
  - プログラム設計（フローチャート）を
    astahにて作成し「ツール」，「画像出力」で
    PNG形式ファイルを作成し
    ”レポートファイル”に貼り付け.
  - 変数表（後述）を”レポートファイル”に貼り付け.
  - テスト仕様書を”レポートファイル”に貼り付け.
  - ソースコード，コンパイル結果，実行結果をscriptコマンドでファイルにし印刷し”レポートファイル”に貼り付け.
- 授業翌週水曜日の午後5時までに
  55号館304室前のレポート提出小箱へ提出

変数表の書き方

- プログラムで使用する変数を表にまとめる

変数名	型	内容	条件
height	int	長方形の高さ	負の数の場合は考えない
width	int	長方形の幅	負の数の場合は考えない

ヒント：課題3-3

- 前判定型の反復処理：　while文
  - 反復を制御する変数を作る
    - 初期値は「小さいほうの整数」
  - whileの条件
    - 反復制御変数と「大きい方の整数」の大小比較
- 合計を格納する変数を作成
  - 初期値は0にしておく必要あり

ヒント：課題3-4

- 方法1：無限ループを使う方法
  - -1が入力されたらbreak
- 方法2：無限ループを使わない方法
  - 入力値が-1以外ならばwhile文を繰り返し
  - 入力値用の変数の初期値に注意

- 2の倍数　⇒　2で割って余りが0になる数
  - 余りを調べるには？

```
if ( x % 2 == 0 )
```

  - 2の倍数であり，同時に５の倍数でもある

```
if ( x % 2 == 0 && x % 5 == 0 )
```

ヒント：課題3-5

- 最高点を求めるには？
  - 「仮の最高点」を格納する変数を作る
  - 「仮の最高点」と「入力された値」を比較
    ⇒　「入力された値」の方が大きい場合には
    「仮の最高点」を書き換える
  - 1人目の場合のみ：
    - 「入力された値」をそのまま「仮の最高点」とする

ヒント：課題3-5

```
#include <stdio.h>
int main(void) {
  int a=5, b=2, c;                    /* 初期化 */
  double x;
  c = a / b;                          /* 計算結果を整数変数に代入 */
  x = a / b;                          /* 計算結果を実数変数に代入 */
  printf("c = %d¥n", c);
  printf("x = %f¥n", x);
  return 0;
}
```

```
c = 2
x = 2.000000
```

X = 2.5にならないことに注意

c = a / b;

x = a / b;

除算を行っている例であるが，答えとなる変数”c”と”x”の型が異なる．

ヒント：課題3-5

計算結果と型（キャスト）

int型変数'a', 'b'をdouble型に変換し，除算を行う

```
#include <stdio.h>
int main(void) {
  int a=5, b=2, c;                    /* 初期化 */
  double x;
  c = a / b;
  x = (double)a / (double)b; /* キャスト */
  printf("c = %d¥n", c);
  printf("x = %.3f¥n", x);
  return 0;
}
```

```
c = 2
x = 2.500
```

ヒント：課題3-6 forの中のfor

```
for (y = 0; y < m; y++) {

    for (x = 0; x < n; x++) {
        printf("#");
    }
    printf("¥n");

}
```

下記の処理をm回繰り返す

n個の#を出力する処理