

プログラミング第一同演習

慶應義塾大学 理工学部 情報工学科

講義担当：河野 健二

演習担当：杉浦 裕太

本日の内容

- C 言語におけるコメント
- 四則演算
- printf() の簡単な使い方
- 変数と代入
 - 整数を扱う int 型の変数
 - 浮動小数点数を扱う float 型の変数
- コンソールからの入力
 - scanf()
- if 文による条件分岐

- プログラム内にコメントを書くことができる
 - `/*` と `*/` で囲む
 - `/*` と `*/` で囲んだ所には自由に何かを書いてよい
- 例:
 - ◆ `/*`
 `* This is a sample program for PR0-1`
 `*/`
 - ◆ `/*` から `*/` まではコメント
- 適宜, コメントを入れる習慣をつけよう
 - 自分のプログラムを1週間後に読むと, 意味不明
 - 大事な場所にはコメントを入れる
 - ◆ 何をどのような考え方で実行しているのか, など

前回のプログラム

- 画面に hello, world と表示するプログラム
 - さしあたって、**青字**の部分は決まり文句として書いておく
 - **赤字**の部分にコンピュータにやらせたいことを書く

プログラムの本体は
ここから始まる。
{と}で本体を囲む

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("hello, world\n");
```

```
    return 0;
```

```
}
```

この 1 文には深い意味がある。
現時点では、最初にこう書いて
おくこと

コンピュータに
やってもらうこ
とを書く

この 1 文には深い
意味がある。
現時点では、最後
にこう書いておく

printf (プリント・エフ) って何？



- 指定した文字列（文字の並びのこと）を表示する
 - `printf("hello, world¥n");`
 - " と " で囲んだ文字列 (hello, world¥n) を画面に表示する

- 例:

```
printf("How are you?¥n");  
printf("I am doing well.¥n");
```

How are you?

I am doing well.

printf (プリント・エフ) って何？



- "hello, world¥n" の ¥n は何を意味するの？
 - そこで改行をしろという意味. ¥n を改行文字という

- 例 (1 行目の ¥n をとったもの):

```
printf("How are you?");  
printf("I am doing well.¥n");
```

How are you?I am doing well.

How are you? の後で改行されず, つながって表示される

- 足し算, 引き算, かけ算は Python と一緒

- $3 + 4$, $5 + 7$, $4 - 1$, $3 - 5$, $3 * 4$, $7 * 3$
- かけ算(と割り算)は先にやる. $3 + 4 * 5$ は 23

- 割り算は / を使う (Python と違う)

- 整数同士の割り算は切り捨てになる
 - ◆ $8 / 2$ (4 になる), $10 / 4$ (2 になる)
- 小数点があるとちゃんとやってくれる
 - ◆ $1.4 / 2$ (0.7 になる), $10 / 2.5$ (4.0 になる)

- 剰余 (割り算の余り) は % を使う

- $14 \% 4$ (2 になる), $10 \% 3$ (1 になる)

四則演算：補足

- $10/4$ を 2.5 と小数まで計算するには？
 - 整数に .0 をつけ小数点のある形にする
 - 次のうち, どの書き方でもよい
 - ◆ $10.0/4$, $10/4.0$, $10.0/4.0$

- べき乗 (Python の `**`) はない
 - べき乗を求めるときは自分でプログラムを書く
 - あるいは, ライブラリ関数を使う
 - ◆ ライブラリ関数とは, よく使う関数をあらかじめ定義しておいたもの

四則演算：上級者向け（初心者は気にしないこと）

■ 負の整数を含む除算と剰余

- a / b は数学的な除算の結果から小数点以下を切り捨てたもの

- 例：

- ◆ $-17 / 3 = -5$ ($-5.666\cdots$ の小数点以下切り捨て)
- ◆ $17 / -3 = -5$
- ◆ $-17 / -3 = 5$ ($5.666\cdots$ の小数点以下切り捨て)

この辺はプログラミング
言語によって違う

- $a \% b = a - (a / b) * b$

- ◆ 難しそうな式だけど、 $17/3$ の余りは $17 - (17/3) * 3 = 17 - 5*3 = 2$ と言っているだけ

- 例：

- ◆ $-17 \% 3 = -2$ $= -17 - (-5) * 3$
- ◆ $17 \% -3 = 2$ $= 17 - (-5) * (-3)$
- ◆ $-17 \% -3 = -2$ $= -17 - 5 * (-3)$

計算結果を表示する (1)

■ 整数を表示するとき:

- `printf("%d", (整数の) 計算式);`
 - ◆ 文字列中の `%d` が計算結果に置き換わって表示される

■ 例:

```
printf("%d", 10 % 3);           /* 改行されないことに注意 */  
printf("%d\n", 7 - 8);  
printf("7 * 7 = %d\n", 7 * 7);
```

1 - 1

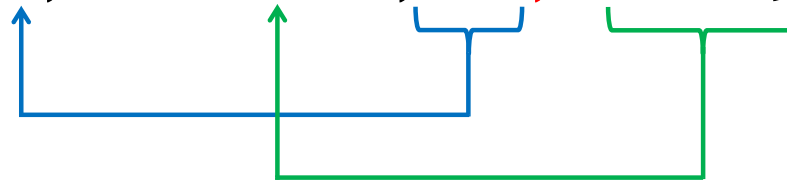
7 * 7 = 49

計算結果を表示する (2)

- 一度にたくさんの結果を表示することもできる
 - 表示したい式をカンマ (,) で区切って並べる
 - 表示したい式の数だけ, %d を " と " の中に書く
 - ◆ %d のところが順に計算結果に置き換わる

- 例:

- `printf("7^2 = %d, 7^3 = %d\n", 7 * 7, 7 * 7 * 7);`



$7^2 = 49, 7^3 = 343$

- 表示結果を予想し, 実行結果と確かめてみよ

```
#include <stdio.h>

int main( )
{
    printf("1 + 2 * 3 + 4 = %d¥n", 1 + 2 * 3 + 4);
    printf("1 * 2 + 3 * 4 = %d¥n", 1 * 2 + 3 * 4);
    printf("1 + 2 / 3 + 4 = %d¥n", 1 + 2 / 3 + 4);
    printf("1 / 2 + 3 / 4 = %d¥n", 1 / 2 + 3 / 4);
    printf("4 + 3 / 2 + 1 = %d¥n", 4 + 3 / 2 + 1);
    printf("4 / 3 + 2 / 4 = %d¥n", 4 / 3 + 2 / 4);
    return 0;
}
```

計算結果を表示する (3)

■ 浮動小数点数を表示するとき:

- `printf("%f", (浮動小数点数の) 計算式);`
 - ◆ 文字列中の `%f` が計算結果に置き換わって表示される
 - ◆ ただし、小数点以下 6 桁まで表示される
 - 表示する桁数は変更できる

■ 例:

- ◆ `printf("%f", 1.1);` → 1.100000
- ◆ `printf("%f¥n", 10 / 2.5);` → 4.000000
- ◆ `printf("Radius: %f, Area: %f ¥n", 3.3, 3.14 * 3.3 * 3.3);`
→ Radius: 3.300000, Area: 34.194600

計算結果を表示する (4)

- 整数と浮動小数点数の混在も可能

- `printf("8 / 3 = %d, 8 / 3.0 = %f\n", 8 / 3, 8 / 3.0);`

`8 / 3 = 2, 8 / 3.0 = 2.666667`

と表示される

- 整数に対応するところは `%d`,
浮動小数点数のところは `%f` にすること！

- 変数を使うには、**まず変数を使うことを宣言**する
 - Python では宣言なしに使うことができる
- 変数を宣言するときは、必ず**型 (type)** を指定する
 - **int** x;
整数を入れるための変数 x を宣言する
 - **float** y;
浮動小数点数を入れる変数 y を宣言する
- 整数と浮動小数点数は**別物扱い**になっていることに注意
 - **整数の 1 と浮動小数点数の 1.0 は別物**と思っていたほうがよい

変数の初期値

- 変数を宣言しただけでは, **デタラメ**な値が入っている

- ```
int x;
printf("x = %d¥n", x);
```

- `x = 0` と表示されることが保証されると思ったら**間違い**

- 変数は必ず**初期化**してから使う

- ```
int x;  
x = 0;  
printf("x = %d¥n", x);
```

- 変数の**初期化忘れ**は致命的な**バグ**

円の面積と円周を計算する例

■ 半径 1.76568 cm の円の面積と円周を計算する

```
#include <stdio.h>

int main( )
{
    float r = 1.76568;
    float pi = 3.14;

    printf("r = %f ¥n", r);
    printf("Enshu = %f ¥n", 2 * pi * r);
    printf("Menseki = %f ¥n", pi * r * r);

    return 0;
}
```

printf で変数の値や、
変数の入った式の値も
表示できる

- 変数には**変数名**をつけて区別する
- 変数名の規則
 - 使用できる文字は, **英字**(`_`を含む), **数字**
 - **先頭**は**英字**
 - 大文字と小文字は区別される
 - 長くてもよい
 - C言語の予約後は使えない (`int`, `float` など)
- わかりやすい変数名を使うようにする

変数の宣言

- 変数の宣言は main の最初で行う
 - 正確なことは、先の講義で教える

- 一般形

型 変数名₁, 変数名₂, ..., 変数名_n;

- タイプ (型) を書き, 変数名をコンマで区切って並べ, 最後にセミコロン(;) を書く

```
int point, subtotal, total;
```

```
int point;  
int subtotal;  
int total;
```

どちらも同じ

代入と代入式

- **代入**: 計算結果などを変数に格納すること
- **代入式**の一般形:

変数 = 式;

- 右辺の式を計算した**結果**を, 左辺の変数に代入する
- **注意: = は数学の等号ではない**
 - 「左辺と右辺が等しい」という**意味ではない**
 - ◆ 右辺の計算結果を変数に代入するという意味

■ 正しい変数宣言は？

- `int mykey;`
- `int MyKey;`
- `int my key;`
- `int my-key;`
- `int my_key;`
- `int _my_key;`
- `int number1;`
- `int 3com;`

- 整数を保持する変数 `i` と `j` を宣言しなさい
- 浮動小数点数を保持する変数 `x` と `y` を宣言しなさい

- 品物の単価, 購入数, 総額を保持する変数を宣言しなさい
 - 例: 単価 100円のを3個購入すれば総額300円

- 長方形の辺の長さを保持する変数を宣言しなさい
 - 例: 長辺は 6.3 cm, 短辺は 3.2 cm

■ 変数 i と j の値は?

```
int i, j;
```

```
i = 3;
```

```
j = 5;
```

```
i = j;
```

■ 変数 i と j の値は?

```
int i, j;
```

```
i = 3;
```

```
i = j;
```

Quiz

- 200分が3時間20分であることを計算する以下のプログラムを完成させなさい。

```
int totalMin, hour, min;  
  
totalMin = 200;  
hour = ;  
min = ;
```

hour に整数 3 が代入され,
min に整数 20 が代入される
ような計算をする

- ヒント:
 - 整数同士の割り算は切り捨て
 - 割り算の余りは % で計算できる

- アメリカでは華氏という単位で気温を表す
 - 世界標準の摂氏とは違う. 摂氏を華氏に変換する
 - ◆ 摂氏 c から華氏 f への変換式: $f = (9 / 5) c + 32$

```
#include <stdio.h>

int main( )
{
    float c, f;
    c = 10.3;
    f = (9.0 / 5) * c + 32;
    printf("%f C is equal to %f F\n", c, f);
    return 0;
}
```

さっきのプログラムは...



- 摂氏 10.3 °C を華氏に変換するプログラム
 - 他の温度, たとえば摂氏 20.0 を華氏に変換するには, いちいちプログラムを書き換える必要がある
 - 面倒くさい...
- キーボードからの入力を受け付けるようにしよう
 - 次のような感じ. 赤字がキーボードからの入力
 - Input Celsius (C): 20.0
20.000000 C is equal to 68.000000 F

キーボードからの入力

■ 整数を読み込むとき

- `int x;` /* 読み込んだ値を入れておく変数 */
`scanf("%d", &x);` /* キーボードから入力した値が変数xに代入される */

■ 浮動小数点数を読み込むとき

- `float x;`
`scanf("%f", &x);`

■ `%d` を使うと整数, `%f` を使うと浮動小数点数

■ 変数の前に付いている `&` を忘れないこと

- `&` の意味はそのうち教える. `&` は C 言語の最大の特徴

■ 入力された摂氏を華氏に変換する

```
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    float c, f;
    printf( "Input Celsius (C): ¥n" );
    scanf( "%f ", &c);
    f = (9.0 / 5)*c + 32;
    printf("%f C is equal to %f F¥n", c, f);
    return 0;
}
```

■ 制御構造とは？

- 変数の値などの条件によって、処理内容を変える仕組み

■ 例：お酒が飲めるかどうか判定するプログラム

- 年齢 (age) が 20 以上なら "OK",
20 未満なら "NO" と表示するプログラム
- 気分としては・・・
 - ◆ もし, age が 20 以上なら `printf("OK¥n");`
 - ◆ もし, age が 20 未満なら `printf("NO¥n");`

C の制御構造: if 文 (1)

- int 型の変数 age に年齢が入っているとする
 - age が 20 以上なら "OK" と表示する
 - age が 20 未満なら "NO" と表示する

```
if (age >= 20) {  
    printf("OK¥n");
```

} ← ここには `;` はつけない

if と (の間,) と { の間に
スペースを入れるとよい

```
if (age < 20) {  
    printf("NO¥n");
```

} ← ここには `;` はつけない

Python と違い, インデントに言語上の意味はない
ただし, 見やすくするために Python のように書くのがよい

C の制御構造: if 文 (2)

- 次の形が一般的な形

```
if (条件) {  
    文1 ;  
    . . .  
    文n ;  
} ← ここには ';' はつけない
```

if と (の間,) と { の間に
スペースを入れるとよい

- 条件が成立するとき, 文₁ から 文_n を実行する
- 条件が成立しないとき, 文₁ から 文_n は実行しない

if 文の条件 (1)

- if 文の条件にはいろいろなことが書ける

演算子	一般形	意味
==	$e_1 == e_2$	e_1 と e_2 の値が等しいなら真, そうでなければ偽
!=	$e_1 != e_2$	e_1 と e_2 の値が等しくないなら真, そうでなければ偽

- 「真」とは条件が成り立つという意味
- 「偽」とは条件が成り立たないという意味

- よくある間違い: 「変数 i の値は 3 と等しいか」

- 誤: `if (i = 3)`
 - ◆ (実は C 言語としては正しい. しかし, 意味が違う)
- 正: `if (i == 3)`

if 文の条件 (2)

- 数の大小関係も調べられる

演算子	一般形	意味
<	$e_1 < e_2$	e_1 の値 < e_2 の値 なら真, そうでなければ偽
>	$e_1 > e_2$	e_1 の値 > e_2 の値 なら真, そうでなければ偽
<=	$e_1 \leq e_2$	e_1 の値 \leq e_2 の値 なら真, そうでなければ偽
>=	$e_1 \geq e_2$	e_1 の値 \geq e_2 の値 なら真, そうでなければ偽

- 以下を表現する関係式を作りなさい。
 - 整数 i が10より大きいか?
 - 整数 i の値は整数 j 以上か?
 - 整数 i と整数 j の積は100以下か?
 - 整数 i から 整数 j を引いた値は正数か?
 - 整数 i は整数 j に10を加えた値以下か?

例題: 標準体重の計算

- 身長 (cm) と体重 (kg) を読み込んで,
標準体重を計算して表示しなさい.
標準体重は次式で求めるものとする
 - $\text{標準体重} = (\text{身長} - 100) \times 0.9$
- 体重が標準体重を 10kg 以上超えていたら,
警告メッセージを表示しなさい

例題: 変数宣言

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main( )
{
```

- ・身長 (height), 体重 (weight), 標準体重 (nrmlweight) は float 型

例題: 身長, 体重の入力

```
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    float height, weight, nrmlweight;

    printf("Enter height: ");
    scanf("%f", &height);

    printf("Enter weight: ");
    scanf("%f", &weight);
```

例題: 標準体重の計算と表示

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main( )
{
```

```
    float height, weight, nrmlweight;
```

```
    /* スライドの都合で入力部分は省略 */
```

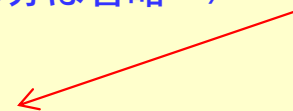
```
    ...
```

```
    nrmlweight = (height - 100.0) * 0.9;
```

```
    printf("height = %f cm, weight = %f kg¥n", height, weight);
```

```
    printf("normal weight = %f kg¥n", nrmlweight);
```

浮動小数点演算であることを
明示する



例題: if 文による条件分岐

```
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    float height, weight, nrmlweight;

    /* スライドの都合で入力部分は省略 */
    ...

    nrmlweight = (height - 100.0) * 0.9;
    printf("height = %f cm, weight = %f kg¥n", height, weight);
    printf("normal weight = %f kg¥n", nrmlweight);

    if (weight - nrmlweight >= 10.0)
        printf("You are overweight!¥n");
    return 0;
}
```

- ・ 体重が標準体重より10kg以上重い場合は警告を表示

if 文の条件(3)

- **論理式**: 2つの関係式や等価式の論理関係を表す式
 - 同時に成り立つか (論理AND演算子)
 - どちらか一方が成り立つか (論理OR演算子)
 - 成り立たないか (論理否定演算子)

名称	演算子	一般形	意味
論理否定	!	!e	e が真なら偽, e が偽なら真. 「e でないなら」という意味
論理AND	&&	e ₁ && e ₂	e ₁ と e ₂ がともに真なら真, そうでなければ偽. 「e ₁ かつ e ₂ 」という意味
論理OR		e ₁ e ₂	e ₁ と e ₂ のどちらかが真なら真, そうでなければ偽. 「e ₁ または e ₂ 」という意味

- 次の条件を表す条件式(論理式)を書きなさい。
 - i が 3 と等しく, かつ j が 4 と等しいか?
 - i が k と等しいか, あるいは j が k と等しいか?
 - i は 2 以上, かつ 10未満か?
 - i は j または k と等しく, かつ x は y または z と等しいか?

ちょっとややこしい話（その1）

- `a == 1 || b == 1 && c == 5` の意味はどっち？
 - `(a == 1 || b == 1) && c == 5`
 - `a == 1 || (b == 1 && c == 5)`
 - ◆ C 言語の規定により, `&&` は `||` より先に処理される
 - ◆ `+` と `*` では `*` が先というのと同じ
 - したがって, `a == 1 || (b == 1 && c == 5)` と解釈される
- 複雑な条件式の場合はカッコをつけて曖昧さをなくそう
 - 慣れればムダなカッコははぶける
 - 慣れないうちはカッコをつけておこう

ちょっとややこしい話（その2）

- $e1 \ \&\& \ e2$ を評価する（真か偽を決める）とき：

- $e1$ が偽であったら $e2$ は評価しない

- 例：

`if (x != 0 && 100 / x == 3) { ... }`

- $x \neq 0$ が成り立たないなら, $100 / x == 3$ は評価しない

- ◆ $x \neq 0$ が偽である時点で, $x \neq 0 \ \&\& \ 100 / x == 3$ 全体は偽
- ◆ 言い換えると, x が 0 のとき $100 / x$ は計算しない

- $x \neq 0$ が成り立つなら, $100 / x == 3$ が成り立つかどうかを調べる

- ◆ $100 / x$ の計算を行い, その結果が 3 に等しいかどうか調べる

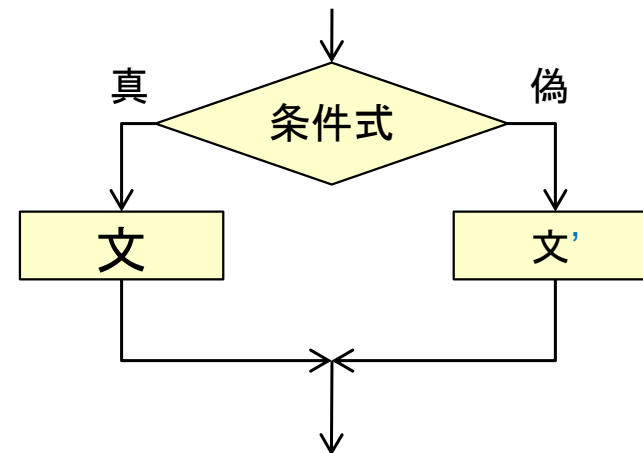
ちょっとややこしい話（その2）

- $e_1 \parallel e_2$ を評価するとき:
 - e_1 が真であったら e_2 は評価しない
- e_1 と e_2 の両者を常に評価するプログラミング言語もある

C の制御構造: if 文 (3)

- 条件を満たすときと満たさないときで実行する文を変えたい場合:

```
if (条件式) {  
    文1;  
    . . .  
    文n;  
  
} else {  
    文'1;  
    . . .  
    文'n;  
  
}
```



条件が偽のとき,
else { ... } の部分が実行される

よくある使い方

```
if (条件1) {  
    文1 ;  
    . . .  
    文n ;  
} else if (条件2) {  
    文'1 ;  
    . . .  
    文'n ;  
} else if (条件3) {  
    文''1 ;  
    . . .  
    文''n ;  
} else {  
    文'''1 ;  
    . . .  
    文'''n ;  
}  
続きの文 ;
```

まず, 条件1 を調べる

- ・ 条件1 が真なら,
文₁...文_n を実行し, 続きの文に進む

条件 1 が偽なら, 条件 2 を調べる

- ・ 条件2 が真なら,
文'₁...文'_n を実行し, 続きの文に進む

条件 2 が偽なら, 条件 3 を調べる

- ・ 条件3 が真なら,
文''₁...文''_n を実行し, 続きの文に進む

条件 3 が偽なら,

文'''₁...文'''_n を実行し, 続きの文に進む

else { ... } の部分はなくてもよい

例題: 標準体重の計算

- 身長と体重を読み込み, その人の標準体重を求め次のようなメッセージを表示しなさい

体重が標準体重に対して,
10%以上超過していれば “overweight”
10%以上少なければ “underweight”
それ以外は “proper weight”

標準体重の計算: 入力と割合の計算

```
#include <stdio.h>

int main()
{
    float height, weight, nrmlweight, rate;

    printf("Enter height: ");
    scanf("%f ", &height);
    printf("Enter weight: ");
    scanf("%f", &weight);

    nrmlweight = (height - 100.0) * 0.9;
    printf("height = %f cm, weight = %f kg¥n", height, weight);
    printf("normal weight = %f kg¥n", nrmlweight);

    rate = (weight - nrmlweight) / nrmlweight;

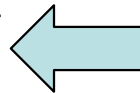
    // continued... ←'//' から行末まではコメント
```


標準体重の計算: 結果の表示

```
// continued...  
  
if (rate >= 0.1) {  
    printf("overweight¥n");  
} else if (rate <= -0.1) {  
    printf("underweight¥n");  
} else {  
    printf("proper weight¥n");  
}  
return 0;  
}
```

・この else は2つ目の if に対応している

```
if (条件) {  
    ...  
} else if (条件) {  
    ...  
} else {  
    ...  
}
```



この形はよく出てくる。
早めに慣れてしまうこと

ちょっと難しいこと: if と else の結びつき

- if と else が入り交じった場合, 両者の対応に注意
 - { と } を使って対応が明確になるようにしよう
 - (見やすくするために) インデントにも注意しよう

```
if (a > b) {  
    if (c > b) {  
        c = a;  
    }  
} else {  
    c = b;  
}
```

else は最初の if に対応する

```
if (a > b) {  
    if (c > b) {  
        c = a;  
    } else {  
        c = b;  
    }  
}
```

else は二つ目の if に対応する

{ と } が省略できる場合もあるが,
その時は if-else の対応に
注意すること

慣れないうちは { と } をつけておく

例題: 閏年の判定

- 4 で割り切れる年は閏年
- しかし, 100 で割り切れる年は平年
- ただし, 400 で割り切れる年は閏年



- 400 で割り切れる年は閏年, そうでなければ
- 100 で割り切れる年は平年, そうでなければ
- 4 で割り切れる年は閏年, そうでなければ平年

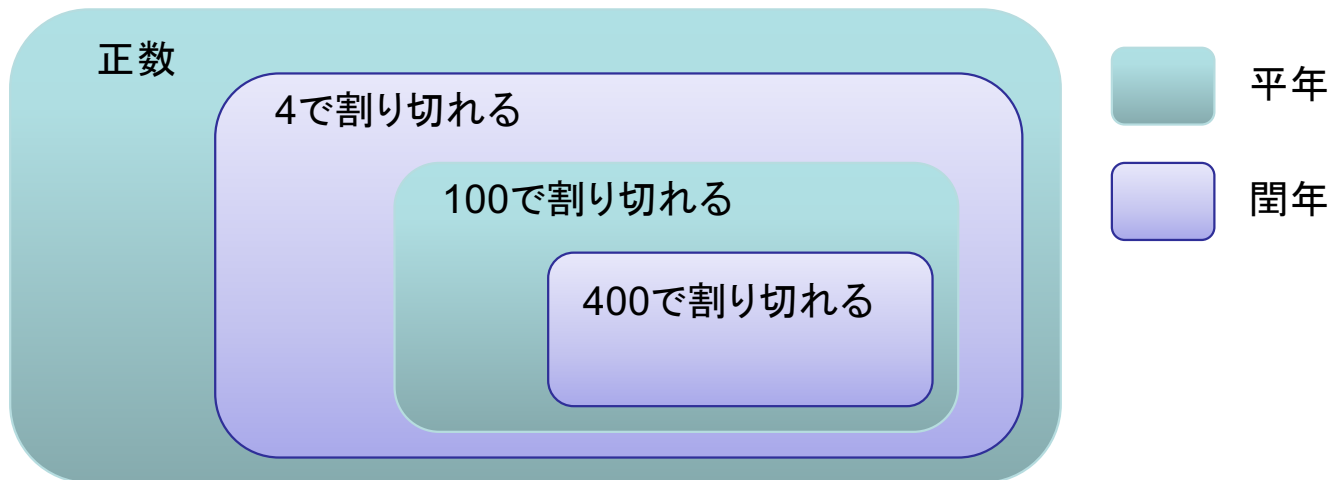
閏年の判定: 解答例その 1

```
#include <stdio.h>

int main()
{
    int year;
    printf("Enter year: ");
    scanf("%d", &year);
    if (year % 400 == 0) {
        printf("%d is a leap year¥n", year);
    } else if (year % 100 == 0) {
        printf("%d is a common year¥n", year);
    } else if (year % 4 == 0) {
        printf("%d is a leap year¥n", year);
    } else {
        printf("%d is a common year¥n", year);
    }
    return 0;
}
```

閏年の判定: 解答例その 2

- 4 で割り切れる年は閏年
- しかし, 100 で割り切れる年は平年
- ただし, 400 で割り切れる年は閏年



閏年の判定: 解答例その 2

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int year;
```

```
    printf("Enter year: ");
```

```
    scanf("%d", &year);
```

```
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
```

```
        printf("%d is a leap year¥n", year);
```

```
    } else {
```

```
        printf("%d is a common year¥n", year);
```

```
    }
```

```
    return 0;
```

```
}
```

4で割り切れるが
100では割り切れない

400で割り切れる

- Python と C は似ているようで, かなり違う
- 四則演算
 - C の / と Python の / と //
- 変数
 - 宣言してから使う
 - 宣言するとき, 型 (type) を指定する
 - 宣言しただけでは, デタラメな値が入っている
- if 文
 - インデントに言語上の意味はない