# ソフトウェア工学 (第1回)

土村 展之
（関西学院大学 理工学部 教育技術職員）
http://ist.ksc.kwansei.ac.jp/~tutimura/

東京農工大学 工学部 情報工学科
2012 年 6 月 23 日





# はじめに



## 自己紹介

- 関西学院大学 理工学部 教育技術職員（2008 年 9 月~ ）
- 「戦略ソフトウェア創造人材養成プログラム」
  （2002 年~ 2005 年：東京大学）
  - 広く使われるソフトを作る人材を育てる
  - ソフトウェア製作の授業も実施
- 研究テーマ：組み合せ最適化・メタ解法・離散凸解析
- TEX 関連ソフト開発
  - ptetex3, ptexlive の開発
  - pTEX の UTF-8 対応拡張
  - xdvi-jp の共同メンテナンス（3 人で）







## 授業のねらい

- C 言語で実際的なプログラムを作れるようになる
  - 関数名・変数名の命名規則
  - 関数への分解
  - 自分で作るか、ライブラリを呼ぶか
- 開発支援ツールを使えるようになる
  - デバッガ、プロファイラ、メモリリーク検出ツール, . . .
- 再現性、移植性の重要性を理解する
- プログラミング上の良い習慣を身につける
  （言語に依存しない）



## 授業の進め方

- 授業は演習形式
- 開講日：6/23(土)，7/7(土)，7/21(土)
- 試験なし
- 毎回レポート課題あり（最終日を含む）

# C言語の基礎知識



## C言語の特徴

- プログラミング言語
  （他に BASIC, COBOL, Pascal, FORTRAN など）
  - コンパイラ型（⇔ インタプリタ型）
  - 手続き型（⇔ 関数型）
- 1972 年頃 D. M. Ritchie 氏と B. W. Kernighan 氏が開発
  （アメリカ AT&T 社 ベル研究所）
- 移植性に配慮
  （「言語規格」と「コンパイラの実装」を分離）
- 言語規模が比較的小さい
- 後の C++, Java, C# 等に大きな影響を及ぼす



## C言語の得意・不得意

- OS のようなシステムよりの記述に向く
  - コンパイル後のプログラムの実行効率を重視
  - 安全なプログラムを書くための配慮が犠牲に
- 標準ライブラリは文字の入出力が中心
- グラフィック表示は言語の規定なし
- 文字列処理は苦手・危険

- 構造化プログラミングをサポート
- オブジェクト指向をサポートしない



## C言語の規格

- K&R (1978 年)
  型チェックなし
- ANSI C (C89, 1989 年)
  関数プロトタイプ, void 型, enum 型
- C99 (1999 年)
  複素数型, 論理型, long long 型, 可変長配列,
  変数宣言がブロックの途中でも可, 1 行コメント (//)
- C11 (2011 年)
  マルチスレッド機能, gets 関数→ gets_s の置き換え

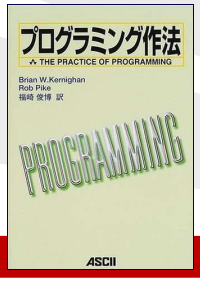
- C99 の機能が全部使えるコンパイラは、普及途中
- 書籍を読むときは、どれを前提としているかに注意
  (K&R の資産は時代遅れ、C99 の解説書はまだ少ない)



## 参考図書

[1] 「プログラミング作法」
B.W. Kernighan, R. Pike 著, 福崎 敏博 訳, アスキー, 2000 年.
（C/C++/Java を対象にした 歴史的な良書。）

[2] 「プログラミング言語 C ANSI 規格準拠（第 2 版）」
B.W. Kernighan, D.M. Ritchie 著, 石田 晴久 翻訳, 共立出版, 1989 年. （バイブル。ただし初学者には不向き。）







## 参考図書

[3] 「新版 明解 C 言語 入門編」
柴田 望洋 著, ソフトバンククリエイティブ, 2004 年.
（scanf() 関数の危険性には注意すること。）

[4] 「C 言語による最新アルゴリズム事典」
奥村 晴彦 著, 技術評論社, 1991 年.

[5] 「C プログラミング診断室」藤原 博文 著, 技術評論社, 2003 年.



# 数値入力



## 課題 (1), (2)

(1) “Hello, world!” を表示しなさい。
- ウォーミングアップ。

(2) ユーザが指定した回数だけ “Hello, world!” を表示しなさい。
- ユーザがどのようにして回数を指定するのか?
- 実現方法は 1 通りではない。
- 回数は int に格納する? double に格納する?



## 解答例 (1)

```
#include <stdio.h>

int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```



## 解答例 (2) その 1

- 回数を コマンドライン引数 から受け取る

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

int main(int argc, char *argv[]) {
    int i, num;

    num = atoi(argv[1]);
    for (i=0; i<num; i++) {
        printf("Hello, world!\n");
    }
    return EXIT_SUCCESS;
}
```

本来は引数なしの場合の処理も必要。



## 解答例 (2) その 2

- 回数を 標準入力 から受け取る

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

int main(int argc, char *argv[]) {
    int i, num;

    scanf("%d", &num);
    for (i=0; i<num; i++) {
        printf("Hello, world!\n");
    }
    return EXIT_SUCCESS;
}
```

本来はエラー処理やプロンプト表示も必要。



## 課題 (3), (4)

(3) 与えられた数値の 2 乗を表示しなさい。
- ユーザがどのようにして数値を指定するのか?
- 実現方法は 1 通りではない。
- 数値は int に格納する? double に格納する?

(4) 与えられた数値の 平方根 を表示しなさい。
- 上の問題と異なる処理は?エラー処理に注意

(4') 与えられた数値の 3 乗根 を表示しなさい。
- どの関数を使う?
- 場合分けは?

## 課題 (5)

(5) 2 つの数値 を受け取り、
それぞれ 2 乗して和を表示しなさい。

- ユーザがどのようにして数値を指定するのか?
- 実現方法は 1 通りではない。
- 将来の拡張を見越して設計してみる。



## 課題 (5')

(5') 5 つの数値 を受け取り、
それぞれ 2 乗して和を表示しなさい。

次の課題は解決すべき問題を多く含んでいるので、次回に。

(6) ユーザの指定する個数の数値 を受け取り、
それぞれ 2 乗して和を表示しなさい。

- 1, 10, 100, 1000 のような個数を想定する。
- ユーザがどのようにして「個数」を指定するのか?
- 受け取った数値を保持する?しない?





# 論理型



## 論理型

- 真偽 (yes/no, true/false) の 2 通りだけを記憶する型
- 変数の型としても、関数の型としても用いられる
- bool 型, boolean 型, 2 値変数, フラグ変数などとも呼ばれる
- TRUE や FALSE は C では直接は用意されていない
  - ANSI C89 なら int 型を用いて
    `#define FALSE 0` などと自分で定義する
  - C99 なら `#include <bool.h>` して、
    bool 型, true, false を用いる



## 論理型の使い方

- if/while 文では TRUE や FALSE と比較しない
  - TRUE や FALSE は変数への代入、関数の返り値に使う
- `flag` という変数名は混乱の元
  - `is_prime`, `has_next`, `visited` なら明らか

```
#include <ctype.h>

if (isalpha(a))  printf("%c is an alphabet.\n", a);
if (isdigit(a))  printf("%c is a number.\n", a);
if (!isdigit(a)) printf("%c is not a number.\n", a);
```



## 課題 (7)

(7) 標準入力から文字を順次受け取り、その文字に対して
`isupper()`, `islower()` 関数などを利用して、
その文字の属性を表示せよ。

```
int main(void) {
    int c;

    while ((c=getchar()) != EOF) {
        ....
    }
    return EXIT_SUCCESS;
}
```

Cygwin では、入力完了は [Enter] の後に [CTRL] を押しながら [D]

## 課題 *(8)*

(8) 標準入力から文字を順次受け取り、その文字に対して `isupper()`, `islower()` 関数などの値を表示し、なぜこのような値になるのか考察せよ。

(8') 異なるコンパイラでの結果を比較せよ。

TRUE との比較が意味をなさない！



## 課題 *(9)*

(9) 1~ 10 の範囲の数字を 2 個 受け取り、
受け取った数に現れなかったものを出力しなさい。

例：2 3 → 1 4 5 6 7 8 9 10

(9') 1~ 10 の範囲の数字を 5 個 受け取り、
受け取った数に現れなかったものを出力しなさい。

(9") 1~ 20 の範囲の数字を 10 個 受け取り、
受け取った数に現れなかったものを出力しなさい。

- フラグ変数や配列の利用を（2 重ループは計算量的に無駄）
- 変数名を工夫する





# 素数計算



## 身につけたいこと

- 小さな機能に分割して関数を分離・独立させる
- 単体テスト

- マジックナンバーの排除
- 配列の扱い



## 課題 *(10)*

(10) 100 未満の素数をすべて表示しなさい。

- 素数とは、2 以上の整数のうち、
1 とその数自身だけで割り切ることのできるもの。
  - 2 → 最初の素数
  - 3 → 2 で割り切れないので素数
  - 4 → 2×2 なので素数でない
  - 5 → 2, 3, 4 で割り切れないので素数



## 解答例 *(10)* その *1*

```
int main(void) {
    int i, j, is_prime;

    for (i=2; i<100; i++) {
        is_prime = TRUE;
        for (j=2; j<i; j++) {
            if (i % j == 0) {
                is_prime = FALSE;
                break;
            }
        }
        if (is_prime) printf("%d\n", i);
    }
    return EXIT_SUCCESS;
}
```

## 解答例 *(10)* その *2*

```
int main(void) {
    int i, j, is_prime;

    for (i=2; i<100; i++) {
        is_prime = TRUE;
        for (j=2; j<i && is_prime; j++) {
            if (i % j == 0) {
                is_prime = FALSE;

            }
        }
        if (is_prime) printf("%d\n", i);
    }
    return EXIT_SUCCESS;
}
```



## 解答例 *(10)* その *3*

```
int is_prime(int n) {
    int i;

    for (i=2; i<n; i++) {
        if (n % i == 0) return FALSE;
    }
    return TRUE;
}

int main(void) {
    for (int i=2; i<100; i++) { // この行は C99 スタイル
        if (is_prime(i)) printf("%d\n", i);
    }
    return EXIT_SUCCESS;
}
```



## 構造化プログラミング

- プログラムを容易に正しく記述することを狙った手法
  - プログラムに階層構造を与える
  - 各階層は 3 種類の制御構造（順次・選択・反復）のどれか
  - 階層毎の独立性を高める
- 1960 年代後半にダイクストラ氏らが提唱
  - “Structured Programming” (1967)
  - “Go to statement considered harmful” (1968)
- 当時のコンピュータ言語にブロック構造がなかった
  - ローカル変数なし
  - サブルーチンを再帰呼び出しできない
  - goto 文でどこへでもジャンプ可能
  - サブルーチンに複数の入口あり



## ループを途中で抜けるなら

1. break
   - ループ 1 段階しか抜けられない
   - 抜けるループとの対応関係が見にくい
   - goto 文の変形 (?)
2. ループ条件にフラグを追加
   - プログラムが複雑に
   - 効率性が少々落ちる
   - 教義的には美しい
3. （推奨）関数に分離して関数の途中で return
   - 独立性が高まる、効率も維持



## 課題 *(10')*, サブルーチン化

(10') 入力された数が素数かどうか判定しなさい。

- 別のプログラムから呼び出せるように、機能を独立させる。

```
int main(int argc, char *argv[]) {
    int num;

    num  = atoi(argv[1]);
    if (is_prime(num)) printf("%d is prime.\n", num);
    else               printf("%d is not prime.\n", num);

    return EXIT_SUCCESS;
}
```



## 課題 *(11)*

(11) エラトステネスの篩 (ふるい) を用いて、100 未満の素数をすべて表示しなさい。

- エラトステネスの篩は、次のような手順で素数を列挙するアルゴリズム。
  1. 調べたい範囲の数字（ここでは 2~ 99）の数字を書き並べる。これが素数の候補となる。
  2. 候補の中から最小の数字を素数として採用する。更に、その素数の倍数を候補から除外する。
  3. 候補が空になれば終了、そうでなければ 2 に戻る。

(11') 1000 万未満の素数の数を数えなさい。

- 巨大な配列を扱うには→ static or malloc()

## 解答例 *(11)*

```
int main(void) {
    int i, j, is_prime[100];

    for (i=2; i<100; i++) is_prime[i] = TRUE;

    for (i=2; i<100; i++) {
        if (is_prime[i]) {
            printf("%d\n", i);
            for (j=i*2; j<100; j+=i) is_prime[j] = FALSE;
        }
    }
    return EXIT_SUCCESS;
}
```



## 課題 *(12)(12')*, サブルーチン化

(12) 入力された数が素数かどうか判定しなさい。
（処理できる数の範囲は自分で設定してよい。）

(12') 和が 100 になる 2 つの素数の組をすべて列挙せよ。

- 別のプログラムから呼び出せるよう、機能を独立させる。
  - 配列に書き込むルーチンと、
    配列参照のルーチンを分離する。
  - 書き込みルーチンを呼び出すタイミングは?



## 解答例 *(12)(12')*

```
int is_prime_flag[100];

void prime_init(void) {
    int i, j;

    is_prime_flag[0] = is_prime_flag[1] = FALSE;
    for (i=2; i<100; i++) is_prime_flag[i] = TRUE;

    for (i=2; i<100; i++) {
        if (is_prime_flag[i]) {
            for (j=i*2; j<100; j+=i) {
                is_prime_flag[j] = FALSE;
            }
        }
    }
}
```



## 解答例 *(12)(12')* 続き

```
int is_prime(int i) {
    if (0 <= i && i < 100) return is_prime_flag[i];
    printf("is_prime: out of range (%d)\n", i);
    exit(1);
}

int main(void) {
    int i;

    prime_init();
    for (i=2; i<=100/2; i++) {
        if (is_prime(i) && is_prime(100-i)) {
            printf("100 = %d + %d\n", i, 100-i);
        }
    }
    return EXIT_SUCCESS;
}
```



## 課題 *(13)*, テスト

(13) 素数判定のプログラムが正しく動いていることを、
第三者に説明したい。そのためのプログラムを書け。

- 効果的な確認ができるよう工夫する。
  - 個々のパーツごとに動作確認（単体テスト）
  - プログラムの全ての部分が実行されるように
  - 境界条件を重点的に
- 「正しいプログラムである」証明は困難ではある。



## 解答例 *(13)*

```
int main(void) {
    int i;

    prime_init();

    for (i=1; i<100; i++) {
        int p1 = is_prime(i);
        int p2 = is_prime2(i);
        if ((p1 && !p2) || (!p1 && p2)) {
            printf("error: i=%d, %d != %d\n", i, p1, p2);
        }
    }
    return EXIT_SUCCESS;
}
```

## 課題 *(14),* マジックナンバー

(14) 和が 100 になる 2 つの素数の組に加え、
和が 98 になる 2 つの素数の組も表示しなさい。

- ソースにおける 100 という数字に 2 種類の意味がある。

- ソースに現れる 100 や 98 という具体的な数字は、
マジックナンバー として嫌われる。
  - その数値の持つ意味がわからない。
  - 数値を変更する場合に、
複数ヶ所を同時に変更せねばならないことが多い。
- 具体的な数字はマジックナンバーとならないよう、
定数や変数に置き換えて、名前をつけるべき。



## 解答例 *(14)*

```
#define PRIME_MAX 100

static int is_prime_flag[PRIME_MAX];

void prime_init(void) {
    int i, j;

    is_prime_flag[0] = is_prime_flag[1] = FALSE;
    for (i=2; i<PRIME_MAX; i++) is_prime_flag[i] = TRUE;
    for (i=2; i<PRIME_MAX; i++) {
        if (is_prime_flag[i]) {
            for (j=i*2; j<PRIME_MAX; j+=i) {
                is_prime_flag[j] = FALSE;
            }
        }
    }
}
```



## 解答例 *(14)* 続き

```
void sum_prime(int n) {
    int i;

    for (i=2; i<=n/2; i++) {
        if (is_prime(i) && is_prime(n-i)) {
            printf("%d = %d + %d\n", n, i, n-i);
        }
    }
}

int main(void) {
    prime_init();
    sum_prime(100);
    sum_prime(98);
    return 0;
}
```





## 暦計算



## 身につけたいこと

- プログラムを作る前に考える
  - ルールを調査する
  - 自作すべきか、ライブラリを探すべきか

- プログラムを自作するとなれば
  - 小さな機能に分割して関数を独立させる
  - 単体テスト
  - 値渡し (call by value) と参照渡し (call by reference)



## 暦計算

- 西暦 y 年 m 月 d 日の前日を求めなさい
  - 2012 年 6 月 23 日 → 6 月 22 日
  - 2012 年 5 月 1 日 → 4 月 30 日
  - 2012 年 4 月 1 日 → 3 月 31 日
  - 2012 年 3 月 1 日 → 2 月 29 日（閏年）
  - 2011 年 3 月 1 日 → 2 月 28 日（平年）

- ルールを調査すべき → 暦は非常に繁雑
- 本来はシステム関数を用いるべき

## グレゴリオ暦の閏年

- 閏年には 2 月が 29 日まである
- 西暦年が 4 で割り切れる年は閏年
  - ただし、西暦年が 100 で割り切れる年は平年
    - ただし、西暦年が 400 で割り切れる年は閏年
- 従って 400 年間に閏年は 97 回
- 日本では 1872 年（明治 5 年）にグレゴリオ暦を採用



## 課題 *(20)*

- 閏年かどうかを判定する関数
  `int is_leap_year(int year)`
  を書きなさい。
- テストプログラムも含めなさい。
  - 平年 2001, 2002, 2003, 2005, 2100, 2200, ...
  - 閏年 2000, 2004, 2008, ...



## 課題 *(21), (22)*

(21) 日付が存在するかどうかを判定する関数
`int is_valid_date(int year, int month, int day)`
を書きなさい。

(22) 1900 年 1 月 1 日を基準に、何日めかを計算する関数
`int date_to_number(int year, int month, int day)`
を書きなさい。
（`date_to_number(1900,1,1)` を 1 とする。）



## 課題 *(23)*

(23) (22) の逆関数
`void number_to_date(int number, int *year, int *month, int *day)`
を書きなさい。

＜呼び出し方の例＞

```
int main() {
    int year, month, day;

    number_to_date(12345, &year, &month, &day);
    printf("%d 年%d 月%d 日\n", year, month, day);
    return 0;
}
```



## 関数が複数の値を返すには

- 大域変数に書き込む。（非推奨）
- 複数の関数に分離して、1 つずつ値を返す。
- 呼出側で準備した変数をポインタで受け取り、書き換える。
  （いわゆる参照渡し）

```
void let_five(int *a) {
    *a = 5;
}
void foo(void) {
    int i = 1;          /* 1 で初期化 */
    let_five(&i);       /* ポインタを渡す */
    printf("%d\n", i);  /* 5 が出力される */
}
```

- 構造体 1 つを返し、中に複数の値を納める。
  （推奨ながらも手間がかかる）



## 課題 *(24)*

- `date_to_number()` と `number_to_date()` を用いて、
  前日の日付を返す `yesterday()` と、
  翌日の日付を返す `tomorrow()` を書きなさい。
  - 引数をどうするかよく考えること。

## 課題 *(25),(25'),* 曜日計算

(25) 1900 年 1 月 1 日が月曜日であることを利用して、
曜日を計算する関数
`int day_of_week(int year, int month, int day)`
を書きなさい。
ただし関数値は 0=日曜, 1=月曜, . . . , 6=土曜を表すとする。

(25') 以下のような出力をするプログラムを作れ。

```
$ cal 6 2012
      June 2012
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```



## ツェラーの公式、ユリウス日

- ツェラーの公式を用いると、曜日計算は以下のように簡単。

```
int day_of_week(int year, int month, int day) {
    if (month < 3) {  year--;  month += 12;  }
    return (year + year / 4 - year / 100 + year / 400
            + (13 * month + 8) / 5 + day) % 7;
}
```

- 経過日の計算には、ユリウス日が便利。
- 通常はシステムの `mktime()`, `localtime()` を用いる。
- 複雑なルールには、自分で対処せず、調べることも重要。



## レポート課題 *(1)*

1. 30 年後の祝日を判定するプログラムは作れない。
その理由を述べよ。

2. 2012 年 6 月 30 日には第 25 回めのうるう秒の挿入が予定されている。 うるう秒への対応状況を OS や環境、あるいは時計などの機器別に調査し、次の点を考察せよ。
   - 対応する意義、しない場合の問題点は何か。
   - 対応していない環境や機器が多いのはなぜか。



## レポート課題 *(2)*

3. 西暦（1900 年以降）と和暦を相互変換する関数群を作れ。
   - 和暦の元号は明治 (=1), 大正 (=2), 昭和 (=3), 平成 (=4) のように数値で表してよいが、
     マジックナンバーとならないようシンボルを定義せよ。
   - 引数は年のみか、月日も必要か、検討せよ。
   - 未来の元号への対処法を考察せよ。
   - テストも含めること。

締切は 7/2(火), 提出先は
`http://ist.ksc.kwansei.ac.jp/~tutimura/tuat/`