

# 情報処理演習

## プログラムの作成とコンパイル

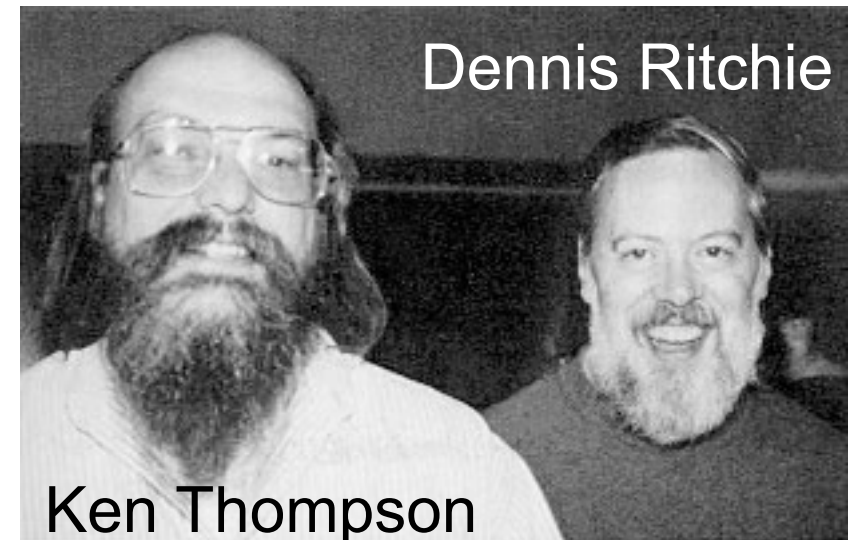
システム科学科 生物工学コース 佐々木耕太

<http://www7.bpe.es.osaka-u.ac.jp/~kota/classes/jse.html>

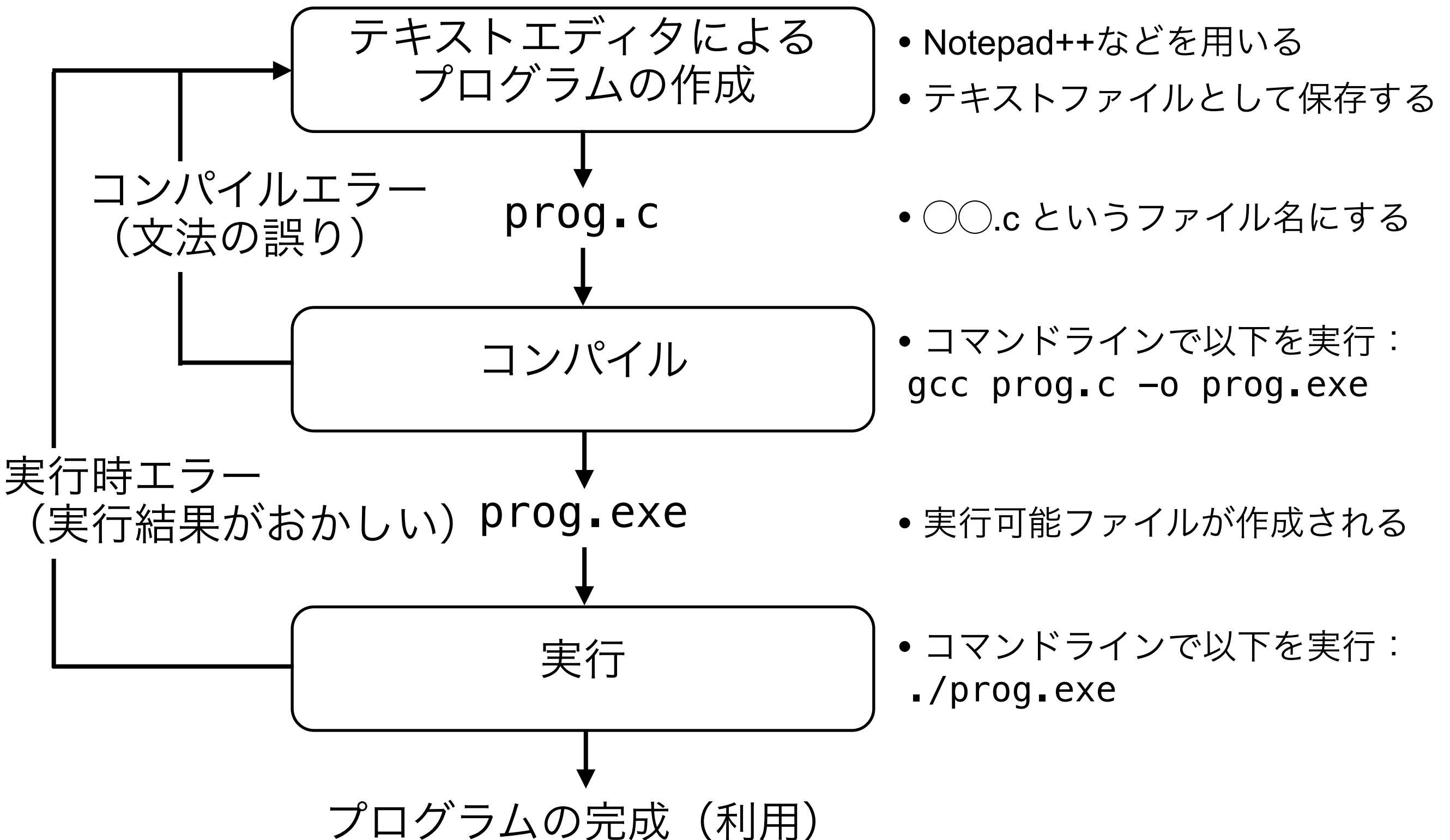
[kota@fbs.osaka-u.ac.jp](mailto:kota@fbs.osaka-u.ac.jp)

# C言語

- 1972年、Dennis Ritchieが開発した
- UNIXとともに発展した  
UNIX(Linuxなど)はすべて  
C言語で書かれている
- 現在、C言語が利用できない環境はほとんどない  
とても多くのソフトウェア製作に利用されている  
家電製品の制御などにも利用されている
- C言語を習得することは、FortranやPascalなどの古くからある手続き型プログラム言語だけではなく、  
C++, Java, Objective-C, Cocoa, Ruby, Pythonなどの  
現代的なオブジェクト指向プログラム言語への入門  
にもなる



# プログラムの作成と検証



# 最も簡単なプログラム

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

.hまで書く  
(.hを省くのはC++の記法)

```
int main(void)  
{
```

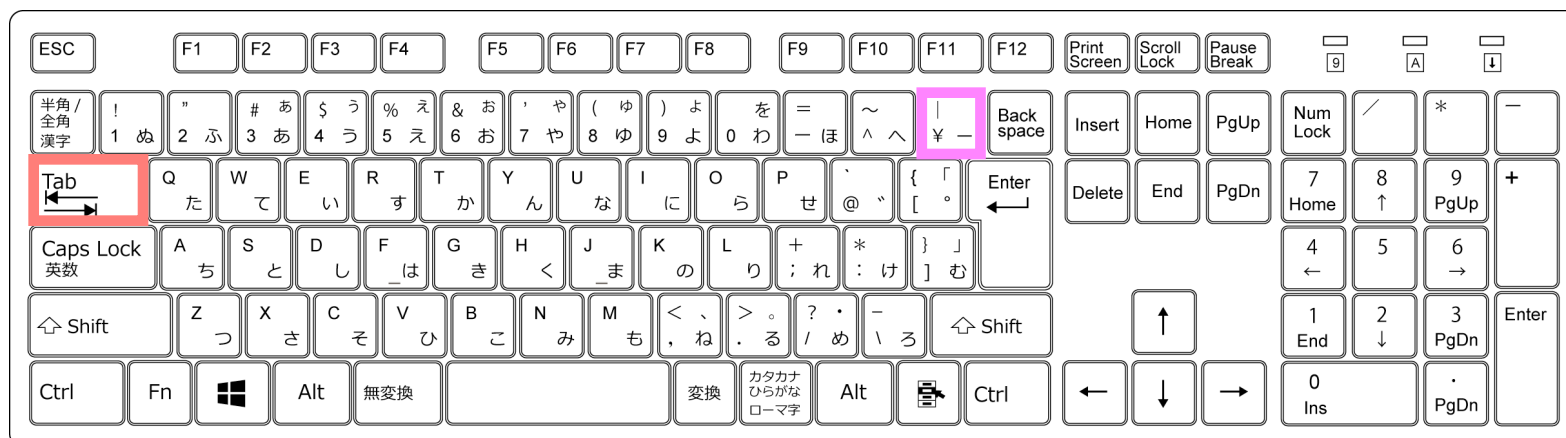
```
    printf("Hello, world!\n");
```

```
    return 0;  
}
```

最後に改行

タブによる字下げ（インデント）  
（字下げされる文字数は環境による）

キーボードの¥で入力する  
（¥と表示されるかもしれない）



以上のプログラムに  
helloworld.cと名前を  
付けて保存する

# プログラムの構造とインデント

- プログラムは `int main(void)` に続く `{` と `}` の間に書く
- `{` と `}` の間には複数の文をおくことができ、それぞれの文は `;` で終わる（途中で改行してもよい）
- `{` と `}` の間にはさらに `{` と `}` をおいて複数の文をひとまとめにすることができる（ブロック）
- ブロックは、より複雑なプログラムを今後作成するとき重要となる
- それぞれのブロックの内側は、タブを入れることでブロックの外側より字下げ（インデント）して見やすくする
- 正しく字下げされていないプログラムは読みづらく、それだけで間違い（バグ）の温床となるので、正しく字下げする習慣をつける

# コンパイル

プログラム（ソースコード）を解釈し、計算機が直接実行することのできるファイルを作ること

```
gcc helloworld.c -o helloworld.exe
```

ソースコードの名前

生成する実行可能ファイルの名前  
(ソースコードと拡張子が違う)

- 以上のgccコマンドを、**ソースコードがあるディレクトリにて**（もしくはソースコードのパスを指定して）実行する
- プログラムに間違い（バグ）がある場合、その行番号とともにエラーの内容が表示される
- `-lm`は三角関数などの数学関数を使用するときに使う（常につけておいてもよい）

例: `gcc calc.c -o calc.exe -lm`

# ソースコードの文字

- 大文字と小文字は区別される（case-sensitive）。
- ダブルクォーテーションの外やコメントの外で、多バイト文字（全角文字）は使用してはいけない。ソースコード中の多バイト文字は、1バイト文字（ASCII文字）がいくつかならんでいるものとしてコンパイラに解釈される。

したがって、例えばソースコード内に全角スペースがあると、おかしいところが見えないのにコンパイルできない。

# コンパイル

ヒトのことは

コンピュータのことは

プログラム

実行可能ファイル

```
#include <stdio.h>

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

```
CAFEBABE00000002000000120000000A
00001000000033C00000000C00000007
0000000300005000000043D80000000C
00000000...
```

コンパイル  
(解釈)

実行

テキスト形式

バイナリ形式





```
sum.c - TeraPad
ファイル(F) 編集(E) 検索(S) 表示(V) ウィンドウ(W) ツール(T) ヘルプ(H)
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int x, y;
6
7     x = 5;
8     z = 6;
9
10    printf("x + y = %d\n", x + y);
11
12    return 0;
13 }
14 [EOF]
```

```
~/linuxhome
The /etc/passwd (and possibly /etc/group) files should be rebuilt.
See the man pages for mkpasswd and mkgroup then, for example, run

mkpasswd -l [-d] > /etc/passwd
mkgroup -l [-d] > /etc/group

Note that the -d switch is necessary for domain users.

u575694g@c-vdi125 ~
$ cd linuxhome
u575694g@c-vdi125 ~/linuxhome
$ gcc sum.c -o sum.exe
sum.c: 関数 'main' 内:
sum.c:8:2: エラー: 'z' が宣言されていません (この関数内での最初の使用)
  z = 6
  ^
sum.c:8:2: 備考: 未宣言の識別子は出現した各関数内で一回のみ報告されます
sum.c:10:2: エラー: expected ';' before 'printf'
  printf("x + y = %d\n", x + y);
  ^
u575694g@c-vdi125 ~/linuxhome
$
```

コンパイルしたら、  
ふたつエラーが起こった

エラーが起こった行の番号




エラーが起こった行に間違いがあるとは限らない  
実際、ふたつめのエラーは2行前の文が  
;で終わっていないことによって起きた

# ファイルの命名についての注意

```
gcc helloworld.c -o helloworld.exe
```

ソースコードの名前

生成する実行可能ファイルの名前  
(ソースコードと拡張子が違う)

-  実行可能ファイルとソースコードの名前を、拡張子まで一緒にすると（せっかく書いた）**ソースコード**が**実行可能ファイル**として上書きされ、**破壊される**
- C言語では、関数の名前を数字で始めたり、関数の名前の途中でハイフンを使うことはできない（ハイフンは引き算をすると解釈される）ので、ソースコードの名前もこうした流儀にならう方が“お行儀がいい”（アンダースコアは関数の名前に含めてもよい）

例： 1-1.c ➡ assignment1\_1.c

アルファベットで始め

ハイフンをアンダースコアに直す

# 標準出力のリダイレクト

標準出力はコマンドラインから変更できる

./helloworld.exe > helloworld.txt

実行ファイルの名前

結果を書き込むファイルの名前

helloworld.exeを実行し、出力はhelloworld.txt  
に書き出す（標準出力をhelloworld.txtに変更する）

# プログラムの構造

```
#include <stdio.h> printfを使うためのおまじない
```

```
int main(void) プログラムはここから始まる（本体は{}の中）  
{
```

```
    int val;    int（整数）型の変数の宣言    変数の宣言
```

```
    val = 5;    値の代入  
    printf("valには%dが代入されました。\\n", val);    出力
```

```
    return 0;    正常に終了させるためのおまじない    実行文
```

```
}
```

%dは（変）数を10進数の（decimal）  
整数に置き換える

# printf関数

- 書式つき (formatted) 文字列を、標準出力に出力する

```
printf("Hello world!\n");  
printf("1 + 2 = %d\n", 3);  
printf("1 + 2 = %d\n", 1 + 2);
```

```
int sum; /* 整数型変数の宣言 */  
sum = 1 + 2;  
printf("1 + 2 = %d\n", sum);
```

- printfを使うためには、プログラムの最初に  
`#include <stdio.h>` と書く

(プリプロセッサ指令で、標準入出力ヘッダーstdio.hを読み込む。stdioはstandard I/O(input/output)の略。studioではない。)

- printfはstdio.hで宣言されている関数のひとつ。関数には、()内で引数を渡す。引数が複数あるときは、,で区切る。

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

```
int main(void)
```

```
{
```

```
    int month, day; /* 整数型変数の宣言 */
```

```
    /* 変数に値を代入する */
```

```
    month = 10;
```

```
    day = 1;
```

```
    /* ... */ コメント
```

他人や未来の自分のために  
わかりやすく書く

```
    /* 結果を出力し、改行する */
```

```
    printf("今日は%d月%d日です。\\n", month, day);
```

printfは2つ以上の値も出力できる

```
    return 0;
```

```
}
```

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

```
int main(void)
{
```

```
    int a,b,c,d,e,f; /* 整数型変数の宣言 */
```

```
    /* 整数の計算をし、結果を変数に代入する */
```

```
    a = 3 + 2 * 4;
```

```
    b = 6 / 4;
```

```
    c = 1 / 2 * 3;
```

```
    d = 12 / (2 * 3);
```

```
    e = a % d;
```

```
    f = 1.2;
```

```
    /* それぞれの結果を出力し、改行する */
```

```
    printf("a = %d \n", a);
```

```
    printf("b = %d \n", b);
```

```
    printf("c = %d \n", c);
```

```
    printf("d = %d \n", d);
```

```
    printf("e = %d \n", e);
```

```
    printf("f = %d \n", f);
```

```
    return 0;
```

```
}
```

/\* ... \*/ コメント

他人や未来の自分のために  
わかりやすく書く

+ 足し算

- 引き算

\* かけ算

/ 割り算の商

% 割り算のあまり



# =は右辺（の計算の結果）を 左辺の変数に代入する演算子



数学の等号とは違う

以下のうち、文法的に間違っている（すなわち、コンパイルエラーとなる）のはどれか？

```
int x, y; /* 整数型変数の宣言 */
```

```
x = 5;
```

```
5 = x;
```

```
y = x + 2;
```

```
x = x + 2;
```

```
x = y = 5;
```

```
x = y = 5 + 2;
```

```
x = y + 2 = 5;
```

```
x + 2 = y = 5;
```