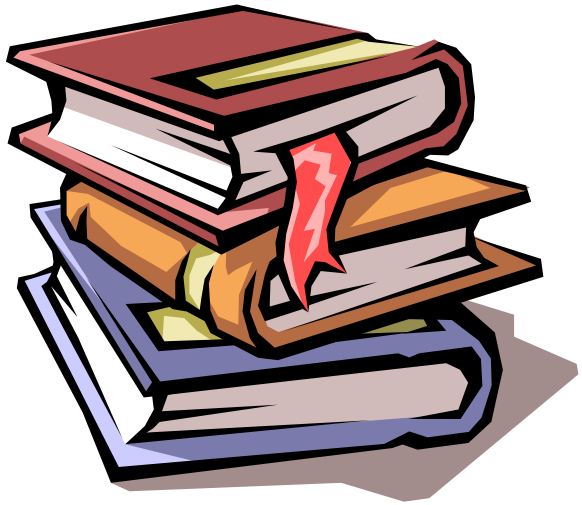
第2回C言語の基本的な規則



今回の目標

- C言語の基本的な規則を理解する。
- C言語のソースコードから実行可能なコードへの変換法を習得する。(コンパイル法の習得)
- 本演習のスタイル規則に慣れる。

☆コンパイル可能で適切に実行できるプログラムを作成する。

世界一短いCのプログラム

shortest.c

```
main(){}
```

このプログラムからわかること

[規則]C言語のソースファイルは、mainという名前の関数が必要。

[規則]括弧が重要（括弧の種類も含めて）

[規則]いろんな部分を省略できる。

実は、コンパイラ任せにしているだけである。
本演習では、省略してはいけない。
スタイル規則参照。

実行ファイルの作り方と実行

(実行ファイルの作り方その1、ガイダンス資料も参照のこと)

ソースファイルから実行ファイルを作ることを「コンパイル」といい、
コンパイルするためのプログラムを「コンパイラ」という。

本演習で用いるコンパイラ

gcc:GNU C Compiler

コンパイラを手動で使う方法

gcc ソースファイル名 -o 実行ファイル名

```
$ gcc shortest.c -o shortest
```

(なお、「ーo 実行ファイル名」を省略すると、
a.outという名前の実行ファイルが生成される。)

プログラムを実行する方法

./実行ファイル名

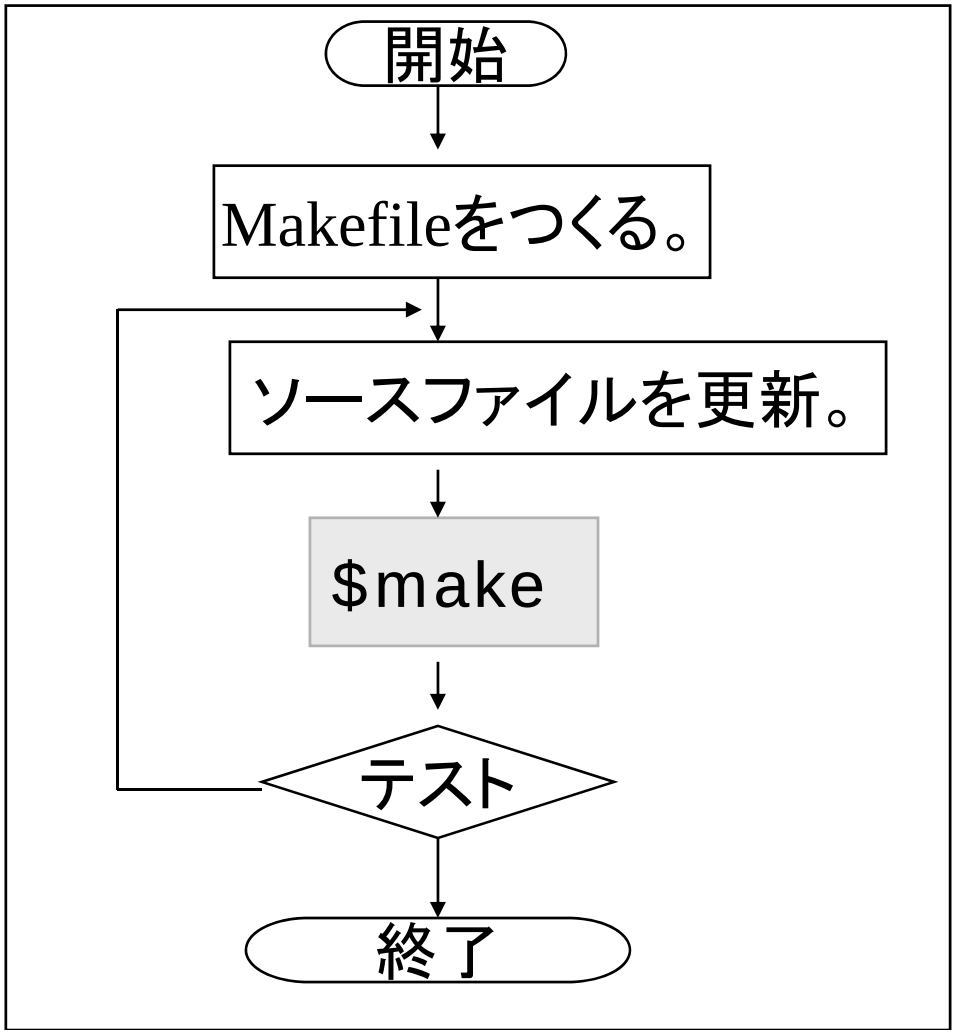
```
$./shortest
```

makeによる実行ファイルの作り方

（実行ファイルの作り方その2、ガイダンス参照)

make:コンパイラを自動で起動するコマンド

makeを使ったプログラミングの流れ。



Makefile:コンパイルのやり方を記述するファイル。本演習ではほとんど同じファイルをずっと使えて便利。

この方法の利点。

・コンパイラへの指示が毎回同じである。
・デバッグ作業が楽にできる。
・間違って、ソースファイルを消す危険性が減る。

Makefile の記述

Makefileには、コンパイラへの指示がいろいろ記述できる。
本演習では、以下のようにかけば良い。

```
CC   = gcc
all: 実行ファイル名（ソースファイル名から「.c」を除いたもの）
```

例

Makefile

```
CC = gcc
all:shortest
```

```
$ make
```

```
$ gcc shortect.c  -o shortest
```

一回Makefileを書くだけで、デバッグごとの
実行ファイルの作成が格段に楽になる。

（もう少し複雑なMakefileは第４回に説明する。）

本演習のスタイルによる最小のソースコード（スタイルA1参照）

```
/*
        作成日: yyyy/mm/dd
        作成者:本荘　太郎
        学籍番号:B0zB0xx
        ソースファイル:tribial.c
        実行ファイル:tribial
        説明:なにもしないプログラム
        入力:なし
        出力:なし
*/
int main()
{
  return  0;
}
```

世界1有名なCのプログラム (教科書p.31)

```
/*
        The most famous program written in C
        hello.c
*/

#include <stdio.h>

main()
{
                printf(“hello ,world ¥n”);
}
```

本演習スタイル版

```
/*
        作成日: yyyy/mm/dd
        作成者:本荘　太郎
        学籍番号:B0zB0xx
        ソースファイル:hello.c
        実行ファイル:hello
        説明:あいさつを表示するプログラム
        入力:なし
        出力:標準出力に「hello,world」と出力する。
*/
#include <stdio.h>

int  main()
{
        printf(“hello ,world ¥n”);
        return  0;
}
```

Cの規則

コメント

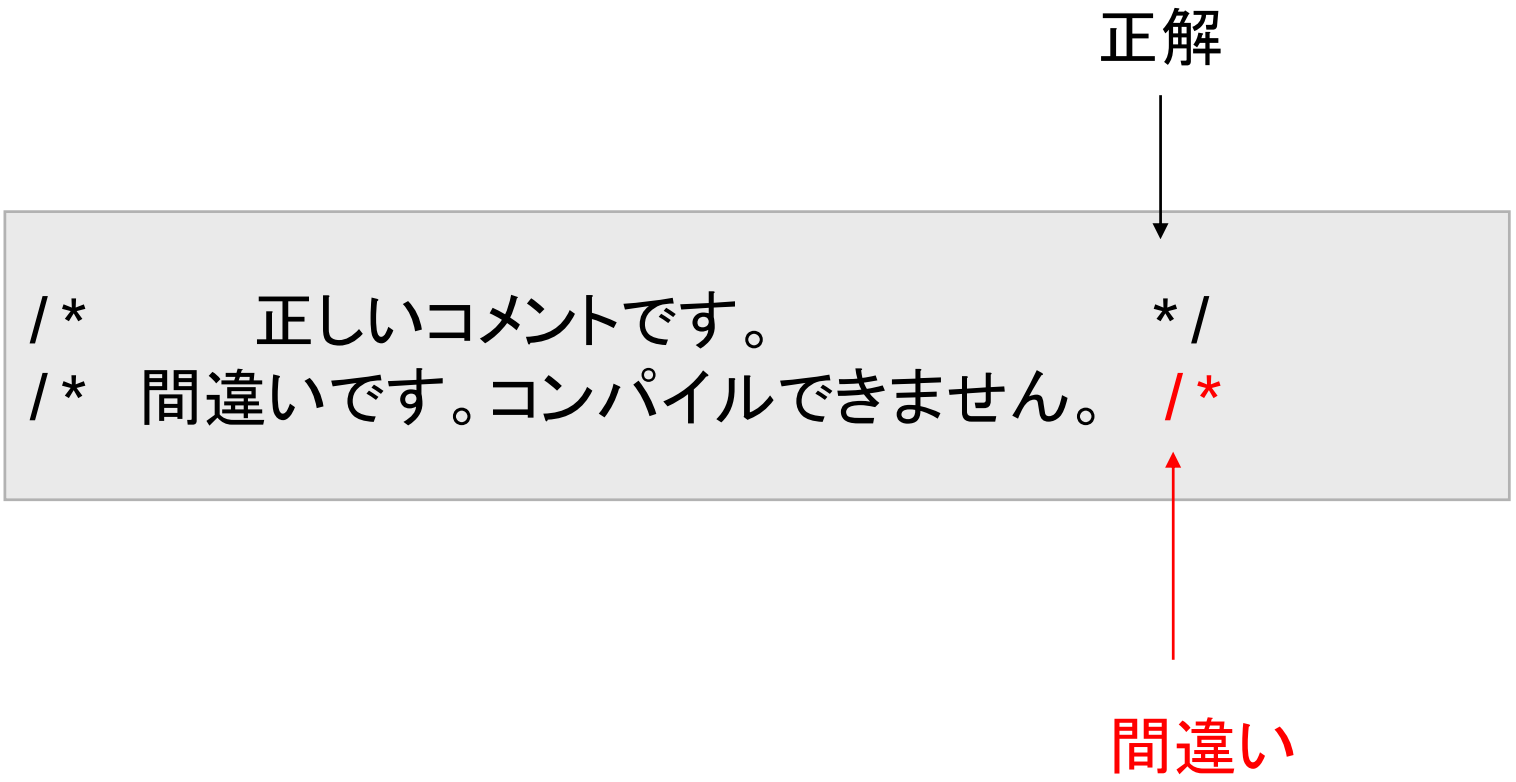
```
/* The most famous program written in C*/
/* hello.c  */
```

[規則] コメントは「/* 」と「*/」で囲む。

いろんなコメント

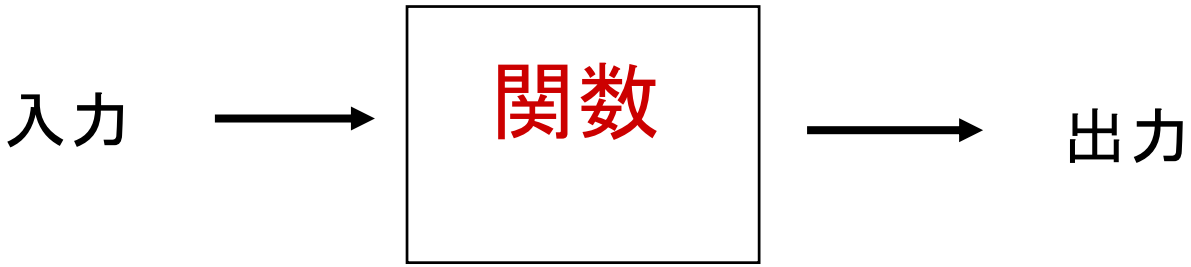
```
/*
課題 T02-1
ename.c
*/

/*****************************/
/*          注目！！！！          */
/*          重要！！！！          */
/*****************************/
```

間違っているコメントの例



関数

[規則] プログラムは関数で構成される。



引数:関数が受け取る入力(例えば実数値)

戻り値: 関数が出す出力(例えば整数値)

関数書式

```
戻り値の型　　関数名(引数の型　引数)
{
    関数の本体
    return 戻り値;
}
```

入力

出力

詳しくは、第9回で説明する。

main関数

プログラム実行時に（必ず）最初に実行される関数。

UNIX系のOSでは、main関数の戻り値型はint型にする。
（教科書では省略されている。）

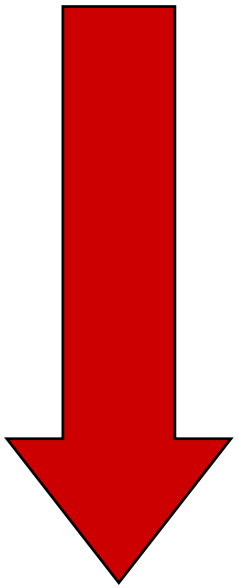
main関数の戻り値型：
本演習では省略しない。

```
int  main()
{
    return  0;
}
```

Unix系のOSでは、
main関数が0を出力することで
正常終了を意味する。

(スタイル規則参照)

C言語でのプログラムの実行順序



```
int main()
{
        * * * * * * *
        * * * * * * * *
        * * * *
        * * * * * * * * *
        * * * * * *
        return  0;

}
```

C言語のプログラムは、
main関数から始まり、
通常は、上から下に、左から右に実行される。

多入力の関数

関数書式

戻り値型

```
戻り値の型　関数名（型1　引数1、型2　引数2、・・・）
{
    関数の本体
    return 戻り値;
}
```

注意：
関数の出力（戻り値）は必ず1つである。

詳しくは、第9回で説明する。

標準入出力

とりあえず、標準入力とはキーボード、標準出力とはディスプレイ（端末）の事だと思うとよい。

注意：関数の入力（引数）や、出力（戻り値）とは無関係なので、注意して下さい。

もし、上記以外を標準入力、標準出力としたいときは、
以下のように リダイレクションを使います。

./実行ファイル名 ＜ 標準入力（ファイル名） ＞ 標準出力（ファイル名）

```
$./hello     >    output
```

```
$lv  output
Hello, world !
```

まだ、入力があるプログラムについては説明しない。
詳しくは次回以降で説明します。

悪いプログラム例1

[規則]文は「;」で終る。

コンパイルできないプログラム

```
#include <stdio.h>

/* 間違ったプログラム*/
 /*By kusakari*/

 #include <stdio.h>

int main()
{
        printf(“hello,world¥n”)
        return  0;
}
```

「;」がないので間違い。
このままでは、
実行ファイルができない。

悪いプログラム例2

[規則]Cには行の概念がない。

```
#include <stdio.h>
/* さっきと同じプログラム*/ /*By Honjo*/ int main(){
printf(“hello,world¥n”);return  0;}
```

このプログラムは、コンパイルはできるが、
読みにくい。上のようなプログラムは、悪い見本。

改行は、適切に行なって、読みやすいプログラムにすること。

複数の命令文と字下げ（インデント）

```
#include <stdio.h>
int main()
{
        printf("hello,");
        printf("world¥n”);
        printf(" By  kusakari¥n");
        return 0;
}
```

1行には一つの命令文。
（長すぎるときは改行して見やすくする。）

インデント

人間がプログラムを読みやすくするための工夫。
中括弧の内部をすべて1タブ分あけてから書く。

(スタイル規則参照）

エスケープ文字

[規則]¥nはエスケープ文字である。

エスケープ文字列集。(教科書p.34)

¥n	改行
¥t	タブ
¥b	バックスペース
¥0	終端文字

¥¥	バックスラッシュ
¥’	シングルクォーテーション
¥“	ダブルクォーテーション

逆の言い方をすると、
¥(バックスラッシュ)で始まる文字列は特別な意味を持つ。
と考えても良い。
なお、この資料では、「¥」でバックスラッシュを表わす。
UNIX上では、「 \ 」がバックスラッシュを表す。

文字の表現（JISコード）

（教科書2章参照。p.19)

[規則]文字を数字で指定できる。

￥＋ ‘8進数’

￥x＋ ‘16進数’

‘A’＝￥201＝￥x41

ソースファイルにおける日本語の扱い

[規則]

日本語（全角文字）は、
コメント内あるいは
printf文の「“」（ダブルクォーテーション）で囲まれた内部
以外で用いないこと。

注意：
特に、全角スペースを上記以外の部分に書かないこと。
正しくコンパイルできない。

プリプロセッサへの指示

[規則]

#で始まる行はプリプロセッサに指示を与える。

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

（教科書7章のプリプロセッサの部分をよく読むこと。）