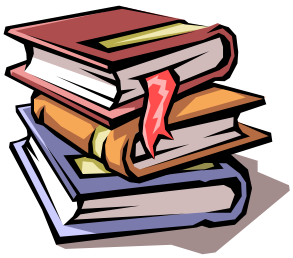
第2回C言語の基本的な規則



1

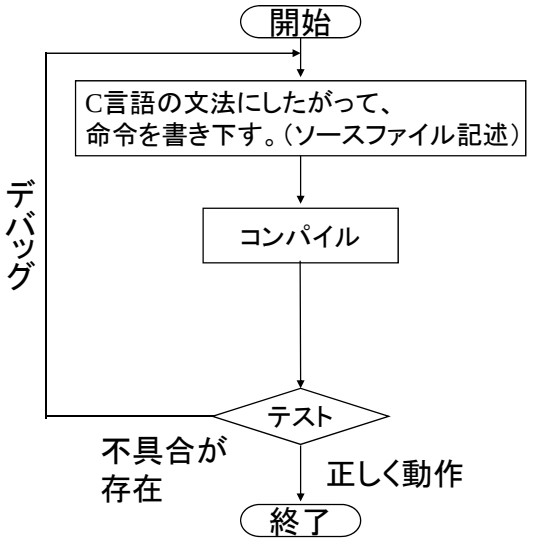
今回の目標

- 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
  実行可能コードへの変換法を習得する。
  （コンパイル法の習得）
- 標準入出力を理解する。
- C言語での標準入出力制御方法を
  理解する。

☆標準入出力を用いたプログラムを
作成する。

2

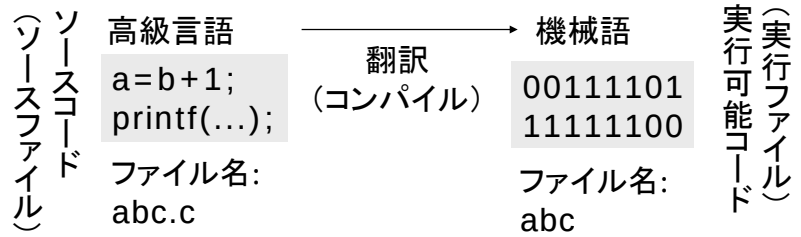
C言語でのプログラム作成手順



使う道具	できるもの
エディタ	ソース ファイル (abc.c)
コンパイラ	実行可能 ファイル (abc)
シェル (OS)	プログラム からの出力

3

コンパイラ



コンパイラ:
高級言語(例えばC言語)から、
低級言語(機械語)へ翻訳(コンパイル)するソフトウェア。

注意:C言語では、ソースファイルは
*****.c
という名前にする。(拡張子が".c"のファイルにする。)

4

プログラム例1（C言語プログラム実行の練習）

```
/*
        作成日: yyyy/mm/dd
        作成者:本荘　太郎
        学籍番号:B00B0xx
        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;
}
```

自分の名前や学籍番号に変更しよう

ソースコードを保存するときのファイル名

この部分を追加する

セミコロン「;」を忘れずに！

実行ファイルの作り方と実行

コンパイル：ソースファイルから実行ファイルを作る作業
コンパイラ：コンパイルを行うためのソフトウェア
　　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  hello.c  -o  hello
```

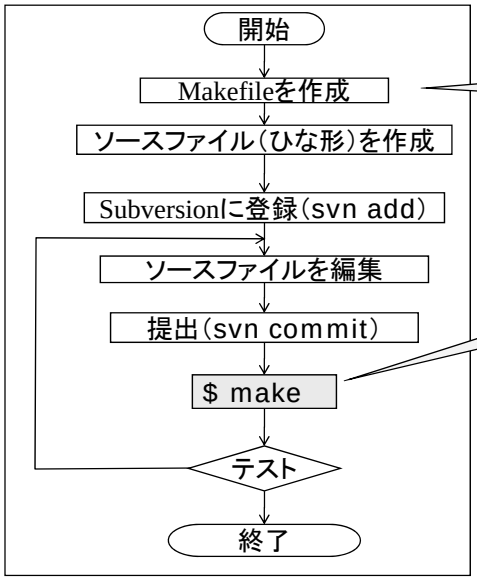
なお、「－o　実行ファイル名」を省略すると、
a.out　という名前の
実行ファイルが生成される。

プログラムを実行する方法

```
./実行ファイル名
```

```
$./hello
```

makeによる実行ファイルの作り方



Makefile : コンパイルのやり方を記述するファイル

make : コンパイラを自動的に起動するコマンド

この方法の利点:

- 手動よりタイプ量が少なくて楽
- 間違ってソースファイルを消す危険性が減る

makeを使ったプログラミングの手順

Makefile の記述

Makefieには、コンパイラへの指示をいろいろ記述できる。
本演習では、以下のように書けば良い。

書式

```
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:hello
```

```
$ hello
```

⇕ 同じ効果

```
$ gcc  hello.c  -o  hello
```

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3回で学習する。）

プログラム例2（標準出力利用の練習）

```
/*
        作成日: yyyy/mm/dd
        作成者:本荘　太郎
        学籍番号:B00B0xx
        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「本荘太郎さん、こんにちは。」と出力する。
*/

#include <stdio.h>

int  main()
{
        printf("本荘太郎さん、こんにちは。¥n");
        return  0;
}
```

この部分を修正し、makeを使って再コンパイル

9

C言語のコメント（プログラムの説明の書き方）

書式

```
/*     この間にプログラムの説明を書く     */
```

スラッシュ＋アスタリスク

アスタリスク＋スラッシュ（順序に注意）

いろんなコメント

```
/*
課題 02-1
ename.c
*/

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

main関数定義

C言語のプログラムは関数定義で構成される。

関数定義　　: プログラムを構成するひとまとまりの処理
main関数定義 : プログラム実行時に（必ず）最初に実行される処理

```
#include <stdio.h>
int  main()
{
        printf("本荘太郎さん、こんにちは。¥n");
        return  0;
}
```

main関数定義の先頭に必ずこのように書く。詳しくは第9回（関数I）で学習する。

処理の最後に return 0; と書く。

括弧を忘れずに

11

プログラムの実行順序

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

}
```

Unix環境下では、
main関数定義に書かれた処理が最初に実行される。
通常は、上から下に、順に実行される。

12

インデント（字下げ）

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。

（Emacs上では、Tabキーで揃えることができる。）

```
#include <stdio.h>
int  main()
{
    →  printf("プログラミング情報表示¥n");
    →  printf("作成者：本荘太郎¥n");
    →  printf("内容：文字列を表示するための、");
    →  printf("練習用コード¥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）

13

標準出力

プログラムを普通に実行したとき：
標準出力＝ディスプレイ（端末）

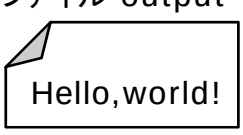
```
$ ./実行ファイル名
```

```
$./hello
Hello,world!
$
```

リダイレクション「＞」を使って実行したとき：
標準出力＝ファイル

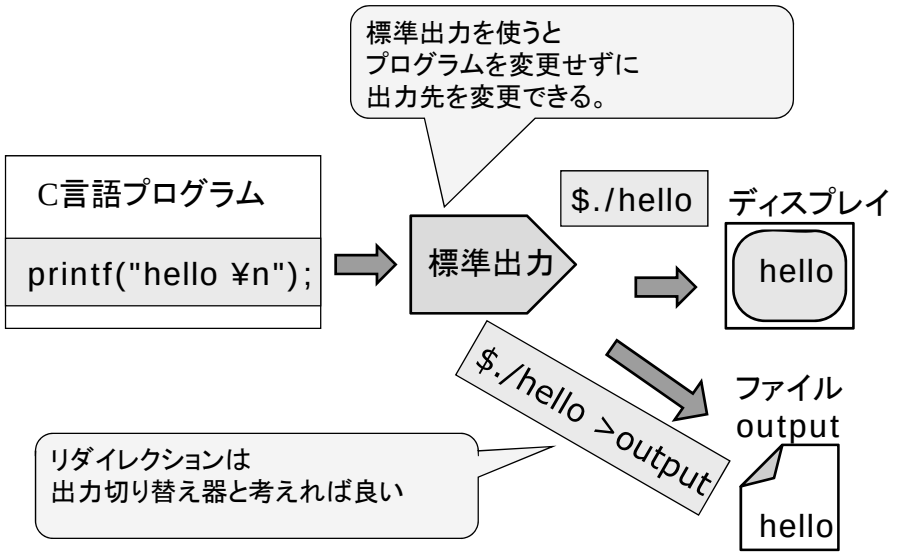
```
$ ./実行ファイル名　＞　ファイル名
```

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



14

標準出力のイメージ



15

エスケープ文字

ソースコード内で¥（バックスラッシュ）で始まる文字列（2文字）は、
コンピュータの内部では一つの記号を表す。
このような文字をエスケープ文字という。

エスケープ文字集

¥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

¥（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。

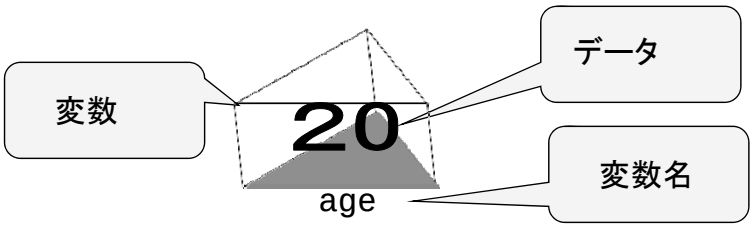
なお、この資料では、「¥」でバックスラッシュを表わす。
UNIX上では、「 ＼ 」がバックスラッシュを表す。

16

変数

変数：データを入れる入れ物

変数名：変数の名前
　　規則にのっとった文字列で命名する。



17

数学とC言語の変数の違い

数学		C言語
入れられるデータ 主に数値で、 整数、実数の区別をしない。	⟺	主に数値、文字で、 種類毎に区別する。 整数と実数も区別する。 （データ型という考え方）
変数名 慣用的に決まっており、 *x, y, t* 等の1文字が多い。	⟺	長い名前を 自分で命名できる。

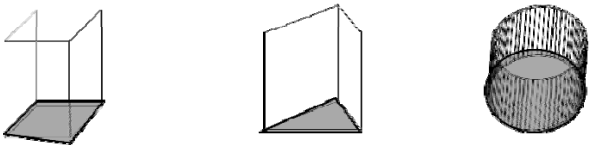
18

C言語の代表的なデータ型

データは、種類ごとに異なる扱いをしなければならない。
データの種類は、データ型として区別される。

char	1バイトの整数型（1つの文字を表す型）
int	整数型
double	倍精度浮動小数点数型

char int double

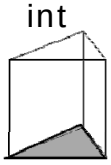


19

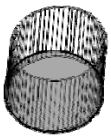
数学の概念とC言語の型の対応

整数	⟷	int
実数	⟷	double
1文字	⟷	char
文字列	⟷	char *

int



double



本演習で主に用いる型:
離散量（年齢・日数など）→ int型
連続量（体重・気温など）→ double型

20

変数宣言

変数宣言によって、プログラムで使う変数を作ることができる。

変数宣言の書式

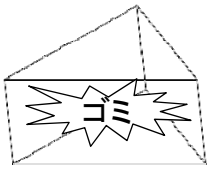
```
データ型1　変数名1;
データ型2　変数名2;
```

変数宣言には、変数の用途
（変数に入れるデータの意味）
を表すコメントを付けること

変数宣言の例

```
int   age;      /*     年齢    */
```

整数値を入れることができる
「age」という名前の変数を作る
変数宣言



age

21

変数宣言の場所

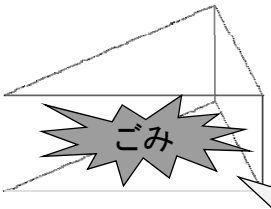
変数宣言は、関数定義の最初でまとめて行う。

典型的なmain関数の定義

```
int    main()
{
       /*変数宣言*/
       int     age;                /*    年齢    */
       double      x_pos;      /*    x座標   */
       double      y_pos;      /*    y座標   */

       ……
}
```

宣言直後の変数の中身

変数の宣言直後では、変数内のデータは不定
プログラムの実行時によって異なる値が入っている



age

変数の中身を常に把握しておくことは、とても重要。
アルゴリズムに従って、適切な値を代入してから
変数を用いることが多い。
このような代入を変数の初期化という。

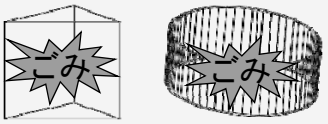
23

変数の初期化

main関数の例

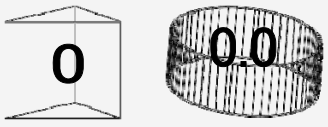
```
int main()
{
   /*変数宣言*/
   int  age;     /*年齢*/
   double  weight;  /*体重*/

   /*変数の初期化*/
   age = 0;
   weight = 0.0;

   /*  ・・・・*/
   ・・・・・・・
```

この段階での状態



age　　weight

この段階での状態



age　　weight

「＝」は、ソースコード内で、
変数に値を代入する方法（代入演算子）。
詳しくは次回学習する。

24

変数の内容の表示

printf文を用いると、
変数に入っているデータを標準出力に出力できる。

- 「"」と「"」の 間の文字列の中に**特別な文字列**を記述
- カンマの後に変数名

変換仕様 :「%変換文字」

printfの典型的な使い方

```
printf("・・・・%変換文字・・・・ ", 変数名);
```

25

printf文と変換仕様

```
printf("あなたは %d 歳ですね。¥n", age);
```

文字列を標準出力(ディスプレイ)に出力するライブラリ関数

変換仕様 printf文の文字列内の「**% 変換文字**」
後ろの変数に関する出力指示を表わす

- 整数

%d 10進数の整数として、int型の値を表示
%6d 10進数として印字、少なくとも6文字幅で表示

- 浮動小数点数(実数)

%f 小数点を使って、double型の値を表示
%6.2f 表示幅として6文字分とり、小数点以下2桁まで表示
%.2f 小数点以下2桁で表示

26

プログラム例3（変数内容表示の練習）

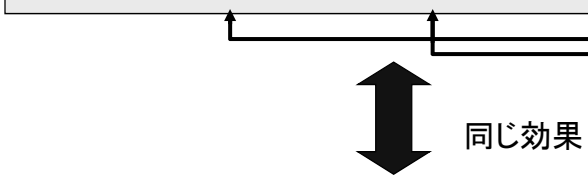
```
/*  変数内容表示練習    print_var.c コメント省略*/
#include<stdio.h>
int  main()
{
        int      a;

        printf("代入前のaの値は%d です。¥n",a);

        a=0;
        printf("0を代入後のaの値は%d です。¥n",a);

        a=3;
        printf("3を代入後のaの値は%d です。¥n",a);

        return  0;
}
```

各行の最後のセミコロン「;」を忘れずに！

printf文における複数の変換仕様

一つのprintf文に変換仕様を複数書いてもよい。

```
printf("first %d  second %d  third %d¥n ",a1,a2,a3);
```



同じ効果

複数のprintf文に分けて書いてもよい。

```
printf("first %d  ", a1);
printf("second %d ", a2);
printf("third %d¥n ", a3);
```

標準入力から変数への値の読み込み

scanf文を用いると、
標準入力（キーボード）から変数に値を代入できる

- 「"」と「"」の間に**特別な文字列だけを**記述
- カンマの後に「**＆変数名**」

変換仕様　：「％変換文字」

scanfの典型的な使い方

```
scanf("％変換文字", ＆変数名);
```

29

scanf文

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

標準入力（キーボード）から変数に値を読み込むライブラリ関数

変換仕様　scanf文の文字列内の「**％変換文字**」
scanfの「"」と「"」の間には**変換仕様しか書かないこと**

＆変数　　scanf文の変数名には&をつけること
&は変数のアドレスを表わす
（詳しくは、第13回で説明する）

- 整数
%d　int型の変数に整数を入力

- 浮動小数点数（実数）
%lf　double型の変数に実数を入力

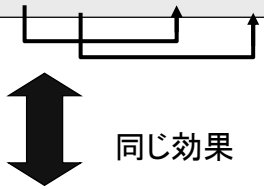
printfの変換文字との使い方の違いに注意！

30

scanf文における複数の変換文字

一つのscanf文に変換仕様を複数書いてもよい。

```
scanf("%d%d", &a1, &a2);
```



同じ効果

複数のscanf文に分けて書いてもよい。

```
scanf("%d", &a1);
scanf("%d", &a2);
```

31

プログラム例4（標準入出力の練習）

```
/*標準入出力練習  test_stdio.c        */

#include<stdio.h>
int main()
{
        int a;
        int b;
        int c;

        scanf("%d%d%d", &a, &b, &c);
        printf("a=%d  b=%d  c=%d ¥n", a, b, c);

        return 0;
}
```

32

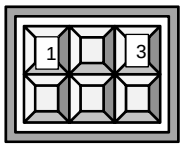
標準入力

注意：標準入力は
関数の入力（引数）とは
無関係。

プログラムを普通に実行したとき：
標準入力＝キーボード（端末）

```
$ ./実行ファイル名
```

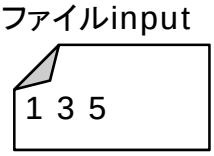
```
$./test_stdio
1 3 5
a=1 b=3 c=5
$
```



リダイレクション「＜」を使って実行したとき：
標準入力＝ファイル

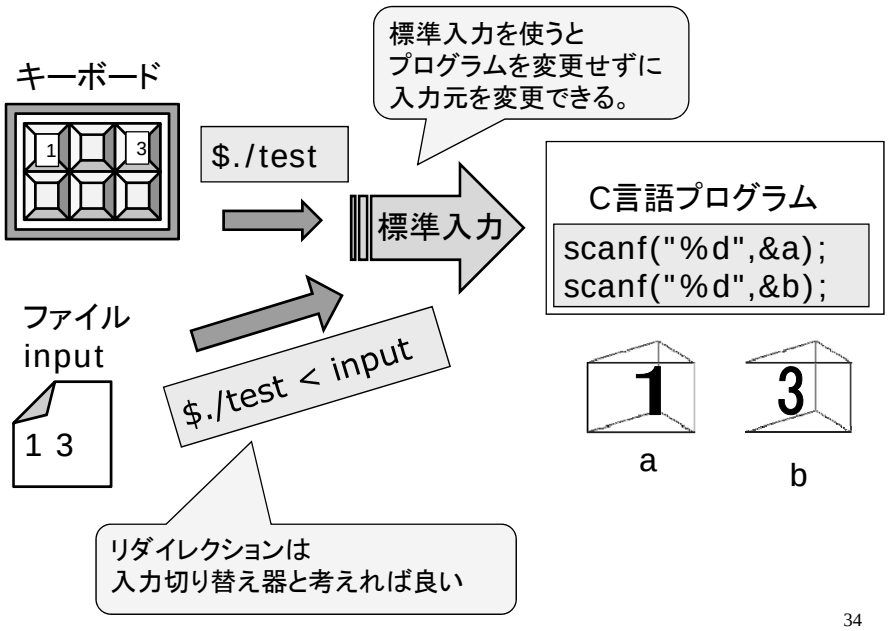
```
$ ./実行ファイル名　＜　入力ファイル名
```

```
$./test_stdio < test_stdio.in
a=1 b=3 c=5
$
```



33

標準入力のイメージ



34

標準入出力

Unixのコマンドは、通常、標準入力と標準出力を用いる。

標準入力：通常はキーボードだが、
リダイレクション「＜」を用いて
ファイルに変更可能。

標準出力：通常は画面だが、
リダイレクション「＞」を用いて
ファイルに変更可能。

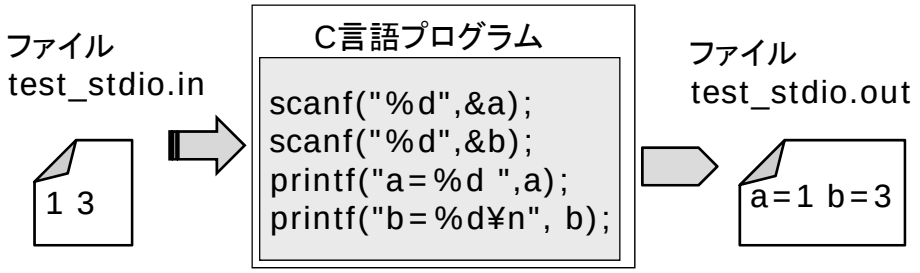
35

標準入出力の同時変更

リダイレクションを組み合わせることで、
入力と出力を同時に切り替えできる

```
$./実行ファイル名　< 入力ファイル　>　出力ファイル
```

```
$./test_stdio  <test_stdio.in  > test_stdio.out
$lv test_stdio.out
a=1 b=3
$
```



36

変数の命名規則

[規則] 名前(変数名、関数名等)は
英字、数字あるいは_(アンダースコア)だけからなり
先頭は数字以外の文字である。

変数名の例

age	x_coordinate
i	y_coordinate
j	x1
k	y1

本演習のスタイル規則:
- 変数の用途(代入されるデータの意味)を反映した名前を付けること
- main関数定義内で宣言する変数は英**小文字**、数字、_(アンダースコア)だけを用い英大文字は用いない事

37

予約語(キーワード)

C言語の予約語

auto	break	case	char
continue	default	do	double
else	for	goto	if
int	long	register	return
short	sizeof	static	struct
switch	typedef	union	unsigned
void	while		

注意:変数名に予約語を用いることはできない。

printf、scanfなど、標準ライブラリ中で利用されている名前も
変数名には利用できない。

38

2バイト文字の扱い

C言語のプログラム中で、
2バイト文字（全角文字）を用いてもよいのは

- コメント内
- printf 文などの「"」と「"」で囲まれた内部

のみ。
それ以外では使うことはできない。

注意：
特に、**全角スペースを書かないように気を付けること。**
正しくコンパイルできない。

プログラム例5：
標準入出力での年齢入出力プログラム

```
/*
        作成日：yyyy/mm/dd
        作成者：本荘　太郎
        学籍番号：B00B0xx
        ソースファイル：echoage.c
        実行ファイル：echoage
        説明：入力された年齢を表示するプログラム
        入力：標準入力から年齢（0以上の整数値）を受け取る。
        出力：標準出力に入力された年齢を出力する。
*/

/*      次のページに続く      */
```

```
/*     前ページのプログラムの続き        */

#include <stdio.h>
int  main()
{
       /*     変数宣言              */
       int    age;        /*入力された年齢*/

       /*     年齢の入力    */
       printf("あなたの年齢は？¥n");
       scanf("%d", &age);

       /*     入力された値をそのまま出力する    */
       printf("あなたの年齢は %d  歳です。¥n", age);
       return  0;
}
```

41

プログラム例5の実行結果

```
$ make
gcc  echoage  -o echoage

$ ./echoage
あなたの年齢は？
20
あなたの年齢は 20  歳です。
$
```

キーボードから打ち込む

42