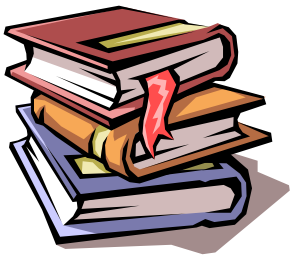
第2回C言語の基本的な規則



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な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

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

[注意1]C言語のソースファイルは、mainという名前の関数が必要。

[注意2]括弧が重要（括弧の種類も含めて）

[注意3]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。
本演習では、省略してはいけない。
スタイル規則参照。

3

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、「－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
```

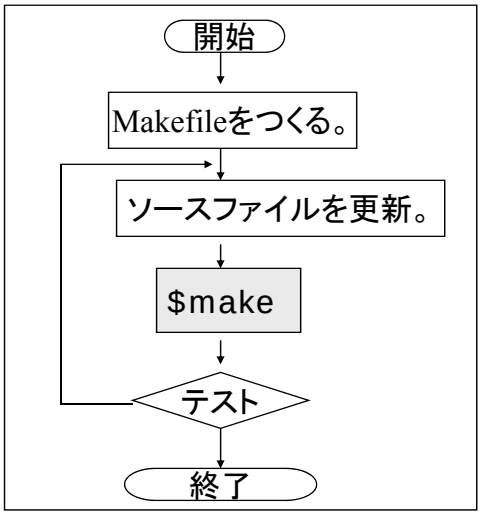
4

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、ガイダンス参照も参照のこと）

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。

5

Makefile の記述

Makefie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。

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:shortest
```

```
$ make
```

⇕ 同じ効果

```
$ 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。）

6

本演習のスタイルによる最小のソースコード

（スタイル規則参照）

```
/*
        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:shortest.c
        実行ファイル:shortest
        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;
}
```

練習1（挨拶表示プログラム）

```
/*
        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: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;
}
```

とりあえず、
このように書く。

C言語のコメント

書式

```
/*     この間にコメントを書く  */
```

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！！！！          */
/*****************************/
```

関数

C言語の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（例えば実数値）

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;
}
```

一種の入力

一種の出力

詳しくは、9～11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。

main関数の戻り値型:
本演習では省略しない。

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

OK

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）

11

プログラムの実行順序

書式を抽出。

```
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、上から下に実行される。

12

インデント（字下げ）

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。
(Emacs上では、Tabキーで揃えることができる。)

OK

```
#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

悪いプログラム例1

NG

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

「;」がないので間違い。
コンパイルできない

NG

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

1行にセミコロンが2個以上ある。
（スタイル規則違反）

14

悪いプログラム例2

```
#include <stdio.h>
int main()
{
        printf(“hello,world¥n”);
}
```

NG

return 文の省略
（スタイル規則違反）

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

NG

インデント不備
（スタイル規則違反）

15

標準出力

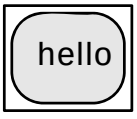
注意：標準出力は
関数の出力（戻り値）とは
無関係。

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!
$
```

モニタ

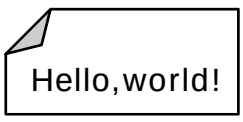


リダイレクション「＞」を使って実行したとき：
標準出力＝ファイル

```
$ ./実行ファイル名　＞　ファイル名
```

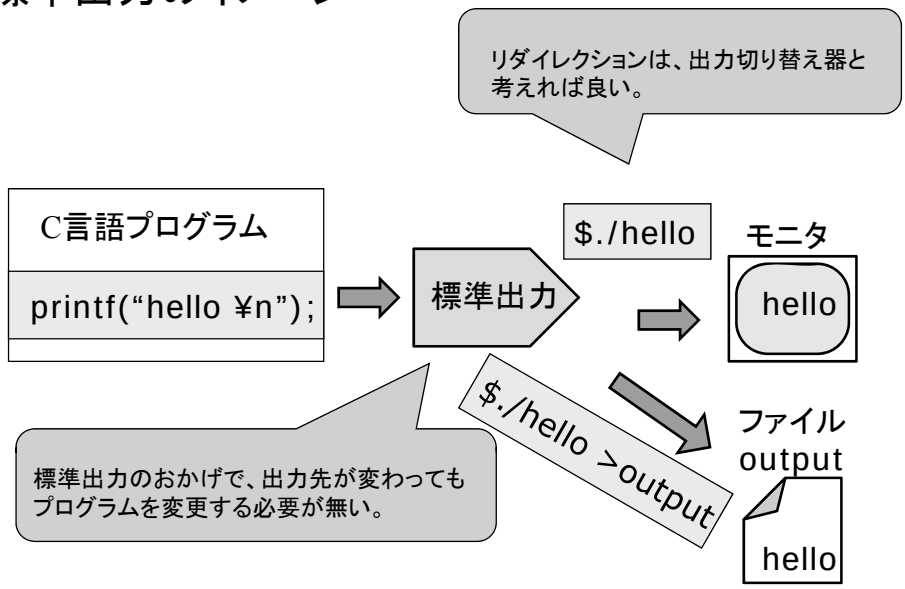
```
$./hello　　>　　output
$
```

ファイル output



16

標準出力のイメージ



17

エスケープ文字

ソース内で¥（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。

エスケープ文字列集

¥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。

18

1バイト文字の表現（ASCIIコード）

1バイト文字（半角文字）を数字で指定できる。

¥x＋16進数

‘A’＝¥x41

上位桁

下位桁	2	3	4	5	6	7
0		0	@	P	`	p
1	!	1	A	Q	a	q
2	“	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	‘	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	[	k	{
C	,	<	L	¥	l	|
D	-	=	M	]	m	}
E	.	>	N	^	n	~
F	/	?	O	_	o	

19

2バイト文字の扱い

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。

20

プリプロセッサへの指示

#で始まる行はプリプロセッサに指示を与える。

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

当面は、プログラム先頭に必ず記入すること。

OK

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

NG

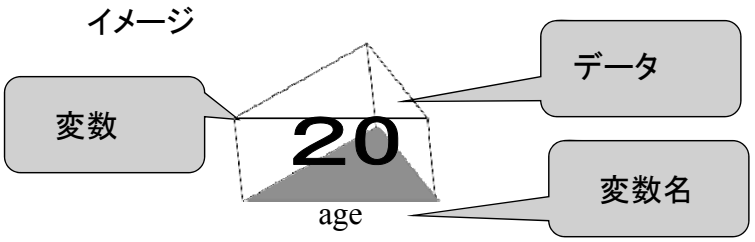
```
int main()
{
        printf("hello,world¥n");
        return  0;
}
```

21

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。



22

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		
主に数で、 整数、実数の区別をしない。	⟺	主に数、文字で、 種類毎に区別する。 整数と実数も区別する。 （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。

23

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。

（スタイル規則参照）

変数名の例

age	x_coordinate
i	y_coordinate
j	x1
k	y1

- なるべく意味のある文字列にする事。
- main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。

24

予約語（キーワード）

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		

C言語の予約語は以上である。

なお、printfとかscanfとかは、ライブラリ中で定義されている。

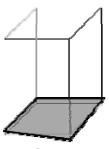
注意：変数名や関数名に予約語を用いてはいけない。
（予約語以外で、命名する。）

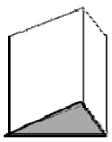
25

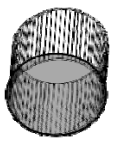
C言語の代表的なデータ型

データは、種類ごとに異なる扱いをしなければならない。
種類は、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
float	単精度浮動小数点型
double	倍精度浮動小数点型


char


int


double

26

C言語のデータ型が扱える範囲

型	バイト長	表現範囲
char	1	0~255
int	4	-214748648~21483647
short int	2	-32768~32767
unsigned int	4	0~4294967295
float	4	$\pm 1.0*10^{-37}$ ~ $\pm 1.0*10^{38}$
double	8	$\pm$ $1.0*10^{-307}$ ~ $\pm 1.0*10^{308}$

27

数学の概念とC言語の型

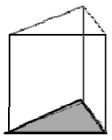
自然数 ⟷ unsigned int

整数 ⟷ int

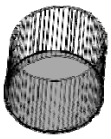
実数 ⟷ float / double

1文字 ⟷ char

文字列 ⟷ char *



int

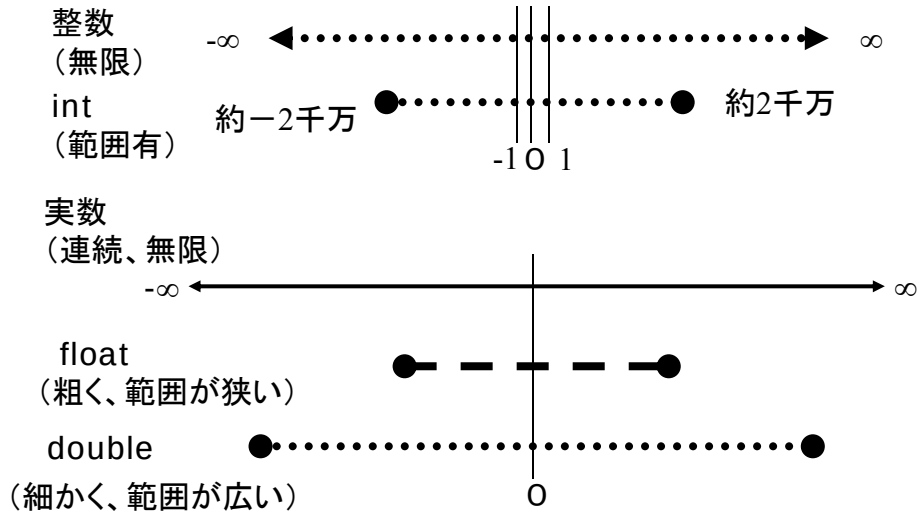


double

本演習で主に用いる型。
離散量→int型
連続量→double型

28

数学とC言語の型の違い



29

本演習で用いる変数の型

（スタイル規則参照）

整数（離散量） int

年齢、
日数、等

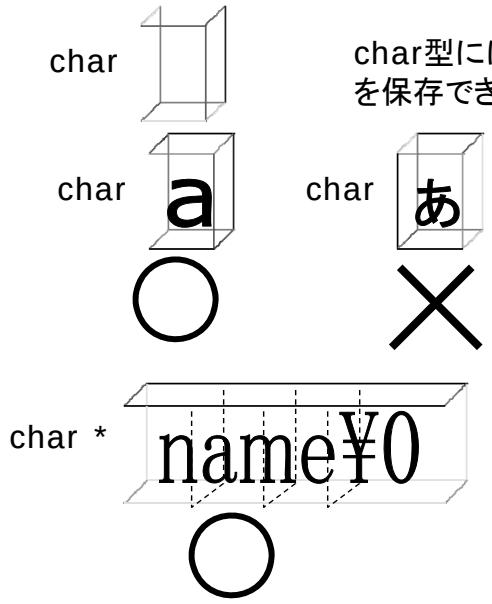
実数（連続量） double

体重、
温度、等

明確に区別する
こと。

30

文字と文字列



31

変数宣言

Cでは使用する変数をすべて宣言しなければならない。

変数宣言書式

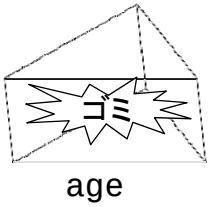
```
データ型1　　変数名1;
データ型2　　変数名2;
```

変数宣言例1

```
int　age;　　/*　　年齢　　*/
```

変数宣言は、プログラム内で用いるデータ
を入れる入れ物（変数）を用意して、
その入れ物に名前をつけると考えればよい。

宣言の際には、必ずコメントを付けること。
（スタイル規則参照）



age

32

変数宣言例2

```
double   x_pos;   /*   x座標     */
double   y_pos;   /*   y座標     */
```

変数宣言例3

```
int   age;          /*   年齢     */
double   x_pos;   /*   x座標     */
double   y_pos;   /*   y座標     */
```

33

変数宣言の場所

変数宣言は、関数の最初でまとめて行う。

典型的なmain関数

OK

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

       ......
}
```

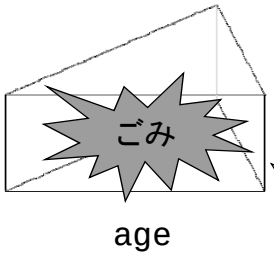
```
int    main()
{
       /*変数宣言1*/
       int  age;    /*   年齢     */

       /*変数宣言以外の処理*/
       f1=0;
       ・・・・・

       /*変数宣言2*/
       double  x_pos;    /* x座標     */
}
```

NG

環境(コンパイラ)によっては、エラーにしない場合がある。
本演習ではできるだけ多くの環境で動作するプログラム作成を目指す。
(コンパイルエラーが無いプログラムが正しいとは限らない。)

宣言直後の変数の中身

変数は宣言しただけでは、変数内のデータは不定です。
つまり、プログラムの実行時によって異なります。

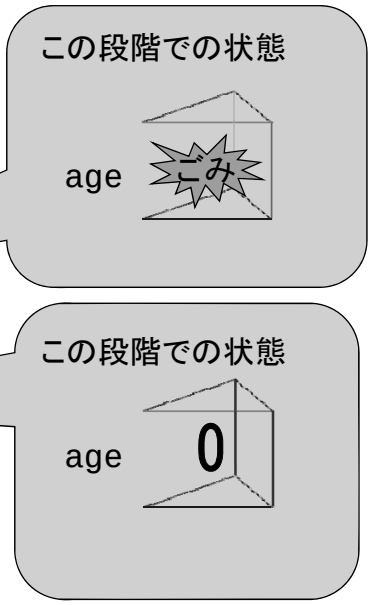


変数の中身を常に把握しておくことは、とても重要。
アルゴリズムに従って、適切な値を代入してから変数を用いることが多い。
このような代入を変数の初期化という。

変数の初期化

典型的なmain関数

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

        /*変数の初期化*/
        age=0;

        /*   ....*/
        .......
```



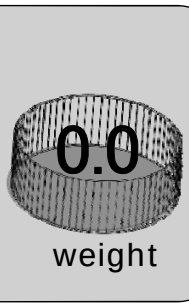
「＝」は、ソースコード内で、
変数に値を代入する方法（代入演算子）。
詳しくは次回。

37

変数の初期化と型

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

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

        /*   ....*/
        .......
```

定数にも型があるので、
注意すること。
詳しくは、次回。
（スタイル規則参照）



38

変数の中身の表示

printf文を用いると、標準出力に変数の中身を出力できる。

- 「”」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);
```

39

printf文と変換仕様

```
printf(“You are %d years old¥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。

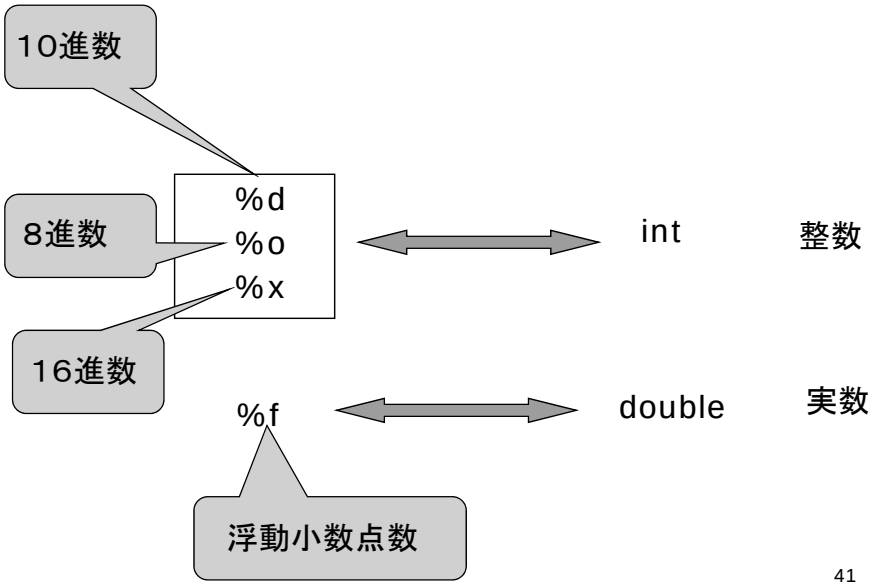
変換仕様 printf文の文字列内の「**%変換文字**」
後ろの変数に関する出力指示を表わす。

(int用)
%d 10進数の整数として表示
%6d 10進数として印字、少なくとも6文字幅で表示
%o 8進数の整数として表示
%x 16進数の整数として表示

(double用)
%f 小数(double)として表示
%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

40

printf文における変換仕様と型の対応



41

printf文における複数の変換仕様

```
int    a1;
int    a2;
int    a3;
a1=1;
a2=2;
a3=3;
printf("first %d  second %d   third %d¥n ",a1,a2,a3);
```

```
int    a;
a=25;
printf("10進数%d 8進数 %o    16進数%x¥n ",a,a,a);
```

同じ変数を
何度も表示

42

printf文に関するよくある間違い

NG

```
int     age;
double      weight;

age=19;
weight=63.5;

printf(“My age is %d ¥n"age);
printf("The weight is %d¥n", weight);
```

カンマ忘れ

（変換仕様と変数の）
型の不一致

43

練習2（変数表示実験）

```
/*  変数の中身表示実験    print_var.c コメント省略*/
#include<stdio.h>
int  main()
{
        int      a;
        printf("代入前 a=%d¥n",a);

        a=0;
        printf("代入後 a=%d¥n",a);

        a=3;
        printf("正しい変換仕様a=%d¥n",a);
        printf("変換仕様間違いa=%f¥n",a);

        return  0;
}
```

44

標準入力から変数への値の読み込み

scanf文を用いて、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**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);
```

45

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。**

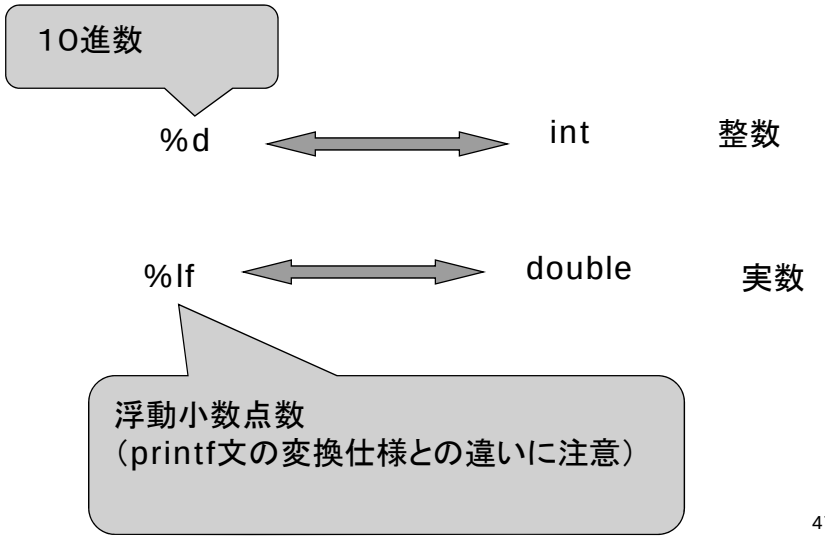
＆変数 scanf文の変数名には&をつけること。
&は変数のアドレスを表わす。
詳しくは、13回ポインタで説明する。

%d 整数を入力
%lf 小数を入力（double型）

printfの変換文字との違いに注意！

46

scanf文における変換仕様と型の対応



47

scanf文における複数の変換文字

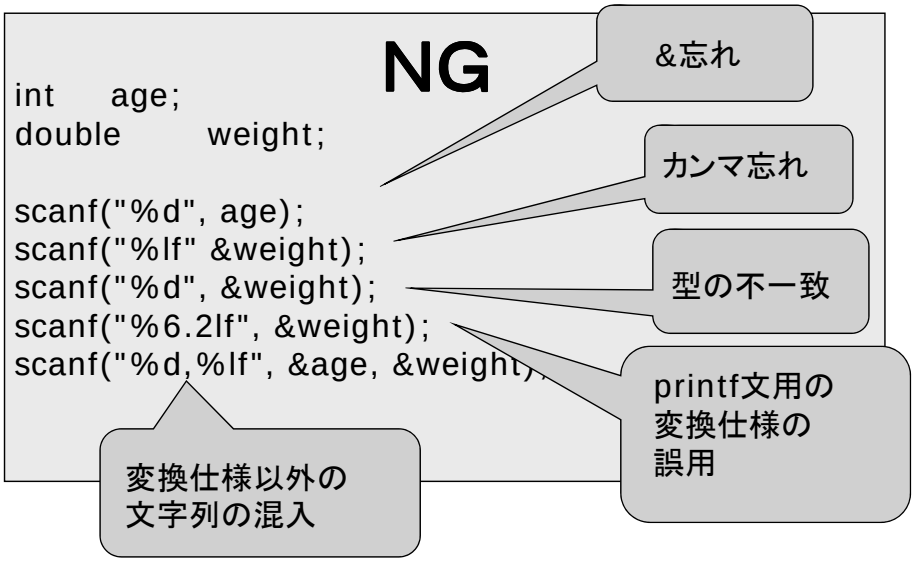
```
char   initial;
int    age;
double  weight;

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

同じ効果

```
/*同じ宣言*/
scanf("%d", &age);
scanf("%lf", &weight);
```

48

scanf文に関するよくある間違い



49

練習3（標準入出力実験）

```
/*標準入出力実験  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;
}
```

50

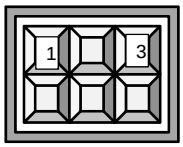
標準入力

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
```

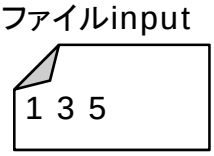
```
$./test_stdio
1 3 5
a=1 b=3 c=5
$
```



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

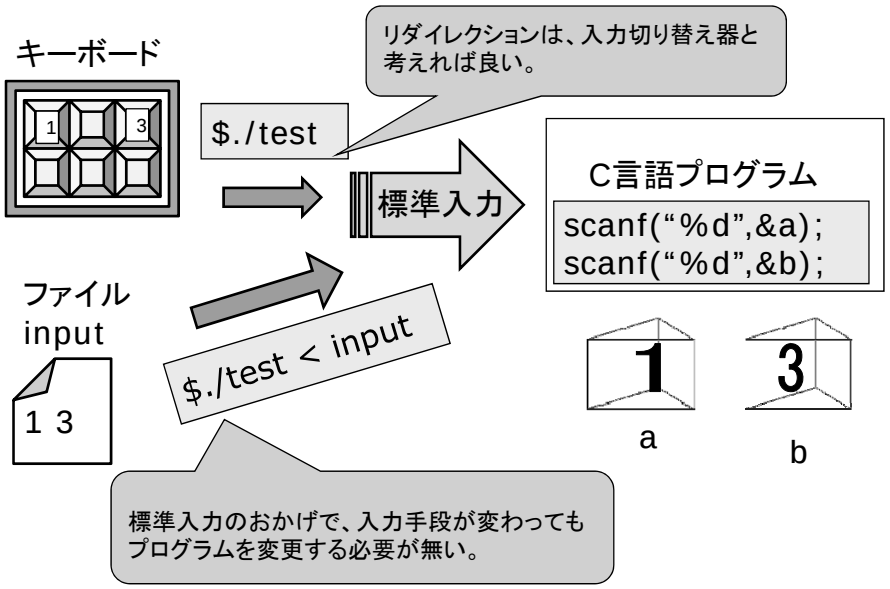
```
$ ./実行ファイル名　＜　入力ファイル名
```

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



51

標準入力のイメージ



52

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。

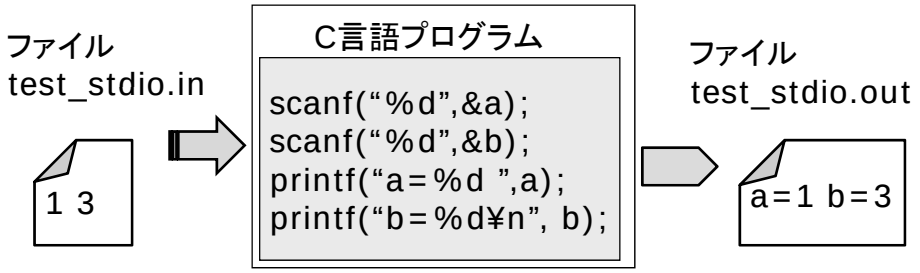
53

標準入出力の同時変更

リダイレクションを組み合わせればいい。

$./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
$
```



54

標準入出力での年齢入出力プログラム

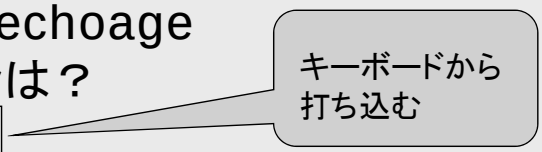
```
/*
        作成日:yyyy/mm/dd
        作成者:本荘　太郎
        学籍番号:B00B0xx
        ソースファイル:echoage.c
        実行ファイル:echoage
        説明:入力された年齢を表示するプログラム
        入力:標準入力から年齢(0以上の整数値)を受け取る。
        出力:標準出力に入力された年齢を出力する。
*/

/*      プログラム本体は次のページ        */
```



```
/*      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("年齢は？¥n");
        scanf("%d", &age);

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

実行結果

```
$ make
gcc  echoage  -o echoage
$ ./echoage
年齢は？
20
年齢は 20  歳です。
$
```



57