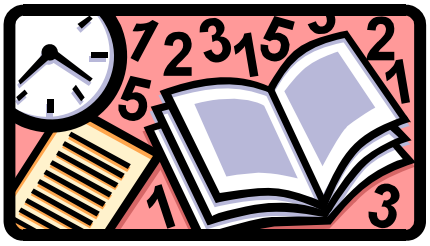
第3回簡単な計算・プリプロセッサ



今回の目標

- 定数とその型を理解する。
- 演算子とその効果を理解する。
- マクロ定義の効果を理解する。
- ライブラリ関数の簡単な使用法を理解する。

☆ヘロンの公式を用いて
3角形の面積を求めるプログラムを
作成する。

定数の分類とデータ型

定数：プログラム中で、常に特定の値を持つ式

定数の種類	表現の対象	データ型	定数の記述例
数値定数	整数	int	123
			0
	浮動小数点数（実数）	double	12.34
			1.0e-5
			0.0
文字定数	1文字（1バイト文字）	char	'a'
			'¥n'
	文字列	char*	"abc"
			"Hello¥n"

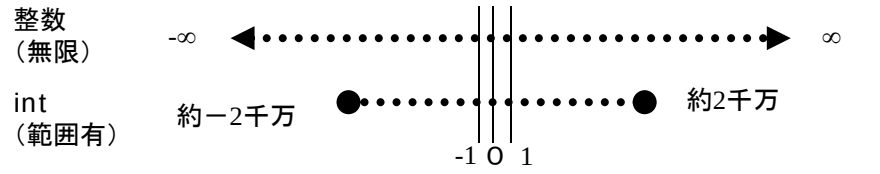
3

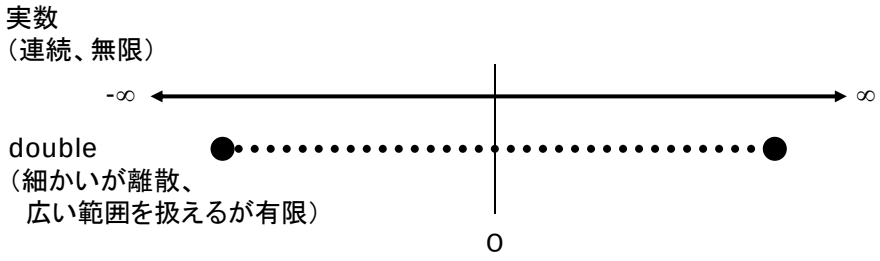
C言語のデータ型が扱える範囲

表現の対象	データ型	データ長（使用するメモリの量）	扱える値の範囲
整数	int	4バイト	-214748648 ~ 21483647
浮動小数点数（実数）	double	8バイト	$\pm 1.0 \times 10^{-307}$ ~ $\pm 1.0 \times 10^{308}$
1文字（1バイト文字）	char	1バイト	0~255

4

数学とC言語の型の違い





5

プログラム（ソースコード）内での数値定数の表現

（整数 ： int型）

（0以外で始まる）数字の列　　123　　123

（浮動小数点数 ： double型）

小数点を含む数字列　　12.34　　12.34

eを使う表現　　1.0e-5　　0.00001

1.0×10^{-5} の意味

6

プログラム例1（数値定数利用の練習）

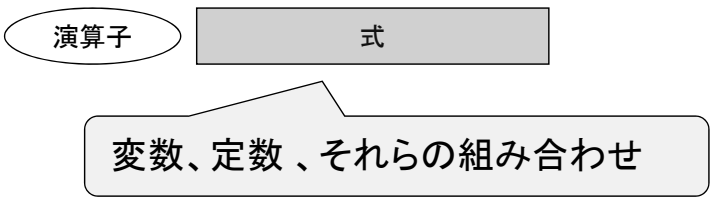
```
/* 数値定数実験  num_const.c  コメント省略    */
#include <stdio.h>
int  main()
{
      int       i;
      double   d1;
      double   d2;

      i = 123;
      d1=12.34;
      d2=1234e-2;

      printf("i    : %8d¥n", i);
      printf("d1 : %8.2f¥n", d1);
      printf("d2 : %8.2f¥n", d2);

      return 0;
}
```

7

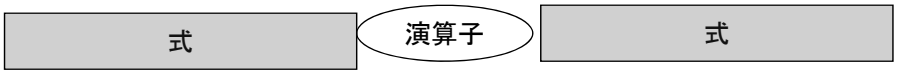
演算子

C言語では、演算子と式（変数、定数等）を組み合わせて、
プログラムが記述される。

単項演算子の書き方（前置型）



二項演算子の書き方



8

算術演算子（C言語での算術計算）

	記号	例	意味
単項演算子	-	-a	aの値と-1の積
2項演算子 （四則演算と剰余演算）	+	a+b	aの値とbの値の和
	-	a-b	〃　差
	*	a*b	〃　積
	/	a/b	〃　商
	%	a%b	aの値をbの値で割ったときの余り

9

数学との表記の違い（1）

	数学	C言語
掛け算	$a \times b$ $a \bullet b$ ab（記号を省略）	edge1*edge2

10

数学との表記の違い（2）

	数学	C言語
指数（べき乗）	a^2	`edge1*edge1`
	$a \wedge 2$	`pow(edge1, 2.0)`

数学ライブラリの関数 pow を利用して
実数のべき乗を計算できる

11

結合規則と細則

[規則] 整数どうしの割り算では、商の小数部分は**切り捨てられる**。

$$\frac{\text{整数}}{\text{整数}} \rightarrow \text{整数}$$

int/int → int

```
int  i;
int  j;
double  x;
i  = 7;
j  = 2;
x =  i / j;
```

上のようなプログラムでは、
x の値は **3.0**である。

```
double      taiseki;
double      takasa;
double      teimen;

taiseki = 1/3 * takasa * teimen;
```

上のようなプログラムでは、
takasa と teimen の値に関わらず、
taiseki の値は **0.0** である。

12

[規則]演算子%は、double型には適用できない。

```
a=b%c;
```

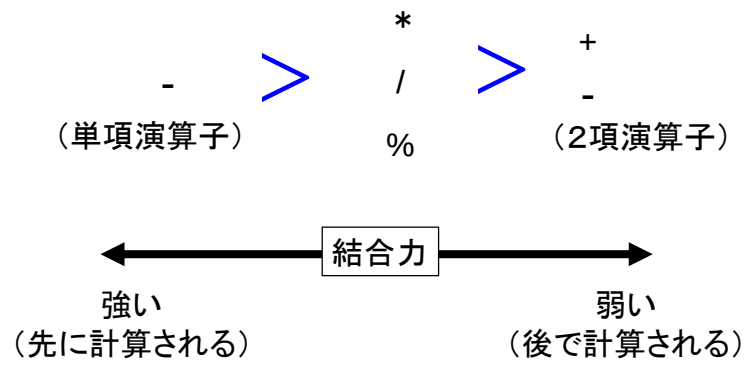
余りを求める演算

このような例では、
a, b, c すべてが整数（int型）でなくてはならない。

例
```
x=7/2;
y=7%2;
```
7÷2は、商が 3 で余りが 1 である。
x　　y

13

演算子の結合力



14

演算子を複数利用した式の計算

結合力が同じ演算子が複数並んだ場合には、
左にある演算子が先に計算される。

`x = a / b *c;` ⟶ x = (a/b) *c

`x = a/(b *c)`とは異なるので注意

`y = -a - b*-c;` ⟶ y = (-a) -(b*(-c))

様々な演算子を使った式では、
括弧をつかって計算の順序を明示した方が良い。

```
x = (a/b) *c;
```

```
y = (-a) -(b*(-c));
```

15

プログラム例2（演算子結合力を確認する練習）

```
/*      結合力確認  priority.c    コメント省略     */
#include <stdio.h>
int  main()
{
        int     a;
        int     b;
        int     c;
        int     d;
        int     x;
        int     y;
        int     z;
        a=5;
        b=4;
        c=3;
        d=2;
        x=0;
        y=0;
        z=0;

/* 続く */
```

16

```
/* 続き */
        x = -a+b/c*d;

        y = -(a+b/c*d);

        z = -((a+b)/c*d);


        printf("x= %3d ,  y=%3d,  z=%3d ¥n",x,y,z);

        return 0;
}
```

17

代入演算子

C言語では「＝」は代入演算子（2項演算子）である。
（数学の「＝」とは違う意味）

書式 変数＝式

- 左辺は必ず変数
- 右辺は式（定数や算術式等）

```
radius = 10.0 ;
```

18

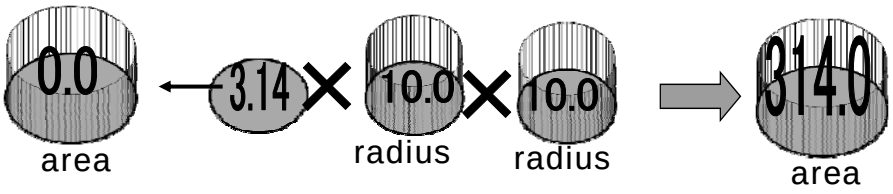
式の計算と代入

C言語では「＝」は代入演算子（2項演算子）である。
（数学の「＝」とは違う意味）

書式 変数＝式

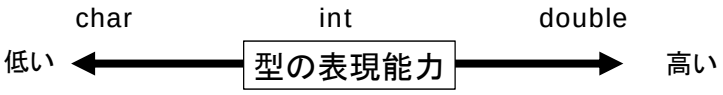
- 代入の右辺は式（定数や算術式等）

```
area = 3.14 * radius * radius ;
```



19

型の表現能力



左辺: double, 右辺 : int → 問題なし

```
double  d;
d = 10 ;
```

左辺 : int , 右辺 : double → 切り捨てが発生

```
int  i;
i = 12.34 ;
```

本演習では、
代入演算子（＝）の左辺の型と右辺の型は必ず同じにすること。

どうしても違う型になるときは、キャスト演算子を用いること。

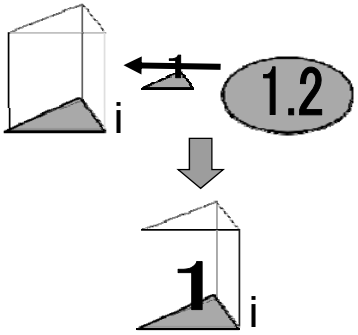
20

キャスト演算子（型の変換法）

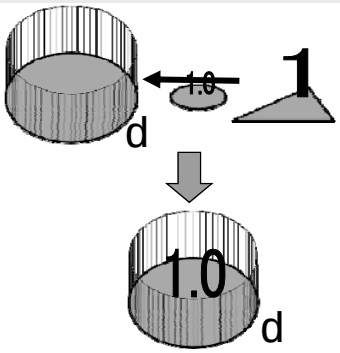
キャスト演算子により、
指定した型に変換した値を求めることができる。

書式　（データ型）式

```
int    i;
i = (int)1.2;
```




```
double  d;
d = (double)1;
```



21

暗黙の型変換

C言語では、異なる型の値同士の演算は、
表現能力が高い型の値に自動的に変換してから行われる。

本演習では、暗黙の型変換を利用せず、
演算子の両辺の式の型を同じにすること。
（必要な場合にはキャスト演算子を利用すること）

```
int   a;
double  b;
double  c;
c = a*b;
```

→

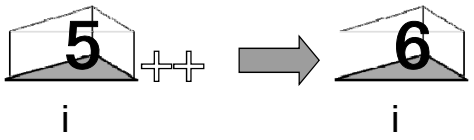
```
int   a;
double  b;
double  c;
c = ((double)a) * b;
```

22

インクリメント演算子とデクリメント演算子

C言語では、1つづつの増減用に、
簡単な形が用意されている。

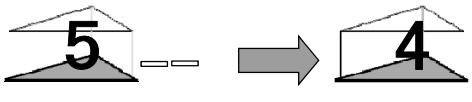
インクリメント演算子　＋＋



別の書き方

i＋＋;　　i=i+1;

デクリメント演算子　--

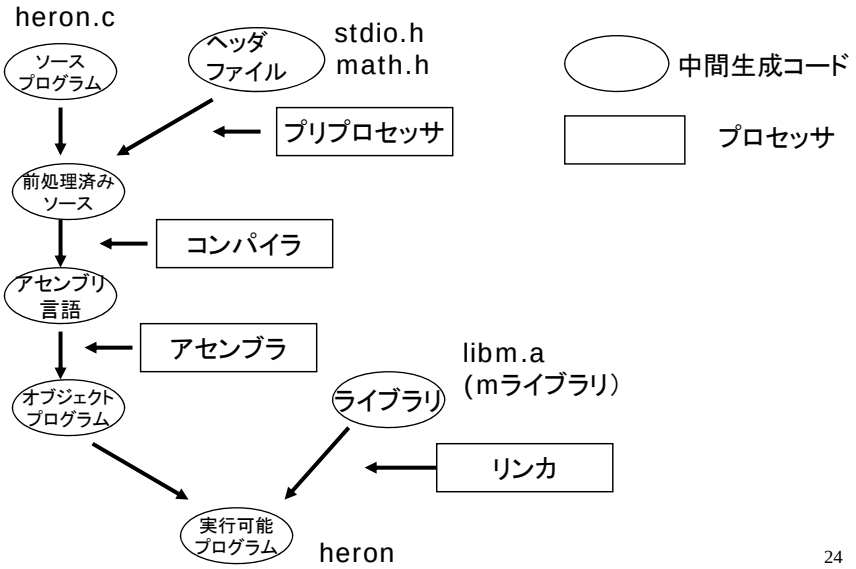


別の書き方

n--;　　n=n-1;

23

gccコマンドの動作



24

プリプロセッサへの指示

[規則]プリプロセッサへの指示行は、
必ず # ではじまる。

例

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

#define  MAX  1000
```

注意：プリプロセッサ行は
行末にセミコロン「;」をつけない。

25

マクロ定義

```
#define　マクロ名　　マクロ展開
```

ソースコードのなかの マクロ名 を マクロ展開 に書き換える

通常の変数と区別するため、
本演習ではマクロ名に英小文字を用いないこと。

例

```
/*  重力加速度  */
#define  G_ACCEL    9.80665
```

マクロ名　　マクロ展開

26

マクロ定義の使用例

```
#define   G_ACCEL   9.80665
int main()
{
   .
   .
   fall = G_ACCEL * time * time / 2.0 ;
```

⇩ プリプロセッサにより
マクロ展開に置き換え

```
int main()
{
   .
   .
   fall = 9.80665 * time * time / 2.0 ;
```

定数（マジックナンバー）を
できるだけ式の中に手作業で
書かないようにする。
間違いの原因になりやすい。

27

プログラム例3（マクロ利用の練習）

```
/*       自由落下距離の導出  fall.c    コメント省略     */
#include <stdio.h>

/*  重力加速度  */
#define  G_ACCEL    9.80665

int  main()
{
        double  time; /* 落下開始後の時間 */
        double  fall; /* 落下した距離 */

        printf("落下開始後の時間(秒):¥n");
        scanf("%lf", &time);

        fall = G_ACCEL * time * time / 2.0 ;

        printf("落下開始後 %8.4f 秒間で、", time );
        printf("%8.4f m落下します。¥n", fall );

        return 0;
}
```

28

ヘッダファイルの利用

他のファイルに書かれたプログラムを
#include でソースファイル内に取り込むことができる。
読み込まれるファイルをヘッダファイルという。

書式

`#include <ファイル名>` → ライブラリに用意されている
ヘッダファイルを取り込む

`#include "ファイル名"` → 自分で作ったヘッダファイル
などを取り込む

29

代表的なヘッダファイル

stdio.h:　標準入出力用
（ printf() , scanf() 等を使うときに必要）

string.h:文字列処理用
（strcpy(), strcmp() 等を使うときに必要）

stdlib.h:数値変換、記憶割り当て用
（atof(), malloc() 等を使うときに必要）

math.h:数学関数（mライブラリ）用
（sin(), cos(), sqrt(), pow()　等を使うときに必要）

30

ライブラリ関数の使い方

書式

関数名（式）

単独で使う場合

関数名（式）;

値を変数に代入する場合

変数＝関数名（式）;

ライブラリ関数：
誰かがあらかじめ作っておいてくれたプログラムの部品。
通常ヘッダファイルと一緒に用いる。
コンパイルオプションが必要なものもある。

詳しくは、第9～11回の関数Ⅰ、Ⅱ、Ⅲで扱う。

31

ライブラリ関数使用例

単独で記述する場合

```
printf("辺1:¥n");
```

（）内の文字列を標準出力に出力するライブラリ関数

他の演算子と組み合わせる場合

```
diag=sqrt(2.0)*edge;
```

同じ効果

sqrt：平方根を求めるライブラリ関数

```
a=2.0;
diag=sqrt(a)*edge;
```

32

Makefile の記述追加

いままでのMakefile

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

数学ライブラリ（mライブラリ）を用いるときは
Makefileにコンパイルオプションを記述

```
CC　= gcc
LDFLAGS＝－lm
all: 実行ファイル名（ソースファイルから.cを除いたもの）
```

33

Makefile 例

```
CC　= gcc
LDFLAGS＝－lm
all: 実行ファイル名（ソースファイルから.cを除いたもの）
```

Makefile

```
CC=gcc
LDFLAGS=-lm
all:heron
```

34

プログラム例4
（ヘッダファイル・ライブラリ関数利用の練習）

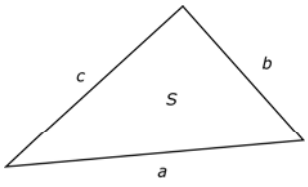
```
/*      球の体積の導出  sphere.c   コメント省略    */
#include <stdio.h>
#include <math.h>


int  main()
{
        double  radius;   /* 球の半径 */
        double  volume;  /* 球の体積 */

        printf("球の半径:¥n");
        scanf("%lf", &radius);

        volume=(4.0/3.0) * M_PI * pow(radius, 3.0);

        printf("半径が %8.4f であるような球の¥n", radius );
        printf("体積は %8.4f です。¥n", volume );

        return 0;
}
```

M_PI:
円周率を表す定数
（math.h内で定義）

pow:
べき乗を求める
ライブラリ関数

35

ヘロンの公式を利用した3角形の面積の導出



ヘロンの公式:
3辺の長さがわかっているときに、
3角形の面積を求める方法。

$$d = \frac{a + b + c}{2}$$

$$S = \sqrt{d\,(d - a)(d - b)(d - c)}$$

36

プログラム例5：3角形の面積を求めるプログラム

```
/*
        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：heron.c
        実行ファイル：heron
        説明：ヘロンの公式を用いて3角形の面積を求めるプログラム
              数学関数を用いるので、-lm のコンパイルオプションが必要。

        入力：標準入力から3辺の長さを入力する。
              各入力は、正の実数とし、どの順序に入力されてもよい。

        出力：与えられた3辺の長さを持つ三角形の面積を標準出力に出力する。
              面積は正の実数であり、小数点以下2桁まで表示する。
*/

/* プログラム本体は次のページ以降       */
```



```
/* 続き */

#include <stdio.h>
#include <math.h>

int  main()
{
        double edge1;        /*辺1の長さ*/
        double edge2;        /*辺2の長さ*/
        double edge3;        /*辺3の長さ*/

        double heron_d;      /* 3角形の周長の1/2 */
                             /* ヘロンの公式で利用 */

        double area;         /*3角形の面積*/

/* 続く */
```

```
/* 続き */

        /* 三角形の辺の長さを入力 */
        printf("辺1の長さ:¥n");
        scanf("%lf",&edge1);

        printf("辺2の長さ:¥n");
        scanf("%lf",&edge2);

        printf("辺3の長さ:¥n");
        scanf("%lf",&edge3);

/* 続く */
```



```
/* 続き */

        /* 3角形の面積の計算(ヘロンの公式) */
        heron_d=(edge1+edge2+edge3)/2.0;

        area=sqrt(heron_d
                *(heron_d-edge1)
                *(heron_d-edge2)
                *(heron_d-edge3)
                );

        /* 計算結果の出力 */
        printf("3角形の面積は  %6.2f  です。¥n",area);

        return 0;
}
```

プログラム例5の実行結果

```
$ make
gcc  heron   -lm   -o  heron

$ ./heron
辺1の長さ:
3.0
辺2の長さ:
4.0
辺3の長さ:
5.0
3角形の面積は       6.00です。
$
```

標準入力
（キーボードから
打ち込む）