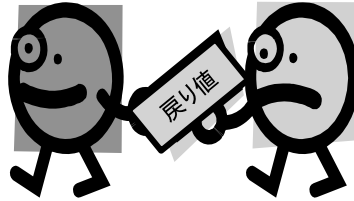


第9回関数 I (簡単な関数の定義と利用)



1

今回の目標

- C言語における関数を理解する。
- 仮引数と実引数の役割について理解する。
- 戻り値について理解する。
- 関数のプロトタイプ宣言を理解する。
- 複数の仮引数を持つ関数を理解する。

☆階乗を求める関数を利用して、組み合わせの数
を求めるプログラムを作成する

2

ライブラリ関数の使い方(復習)

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。

3

ライブラリ関数使用例

単文として記述する関数

```
printf("辺1:¥n");
```

printf: 指定された文字列を標準出力に出力するライブラリ関数

式の中で使う関数

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

sqrt: 平方根を求めるライブラリ関数

今回は、ライブラリ関数 sqrt などと同じように、式の中で使うことができるような関数を自分で作る方法、使う方法について学ぶ。

4

標準ライブラリの数学関数(一部)

$\sin(x)$	$\sin x$: x の正弦
$\cos(x)$	$\cos x$: x の余弦
$\tan(x)$	$\tan x$: x の正接
$\log(x)$	$\log_e x$: x の(自然)対数
$\exp(x)$	e^x : e の x 乗(e は自然対数の底)
$\text{sqrt}(x)$	$\sqrt{x}$: x の平方根
$\text{pow}(x, y)$	x^y : x の y 乗
$\text{fabs}(x)$	$ x $: x の絶対値

x や y は
double型の式

ヘッダファイル読み込み(プログラム先頭で)
#include <math.h>

コンパイル時(Makefile) mライブラリの指定
LD_FLAGS = -lm

5

プログラム例1: 数学関数利用の練習

```

/* 数学関数利用の練習 hypo1.c コメント省略 */
/* 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 */
#include <stdio.h>
#include <math.h>
int main()
{
    double base; /* 直角三角形の底辺の幅 */
    double height; /* 直角三角形の高さ */
    double hypo; /* 直角三角形の斜辺の長さ */

    printf("底辺 ? ￥n");
    scanf("%lf", &base);
    printf("高さ ? ￥n");
    scanf("%lf", &height);

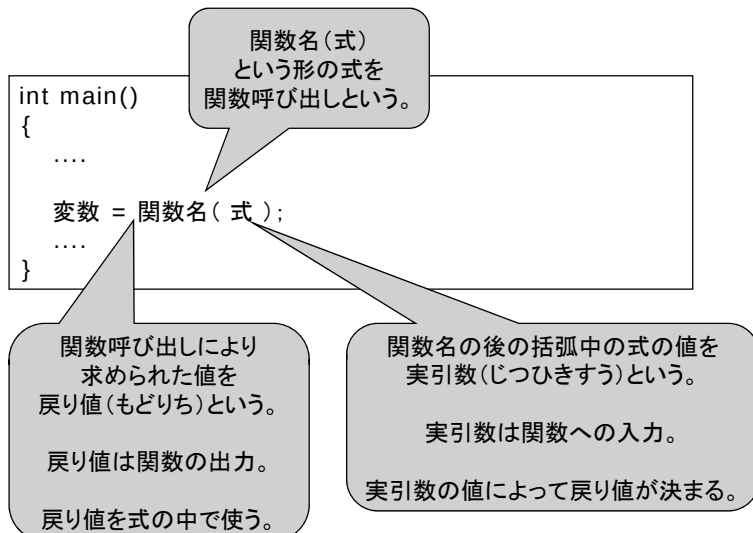
    hypo = sqrt( pow(base, 2.0) + pow(height, 2.0) );

    printf("底辺:%f , 高さ:%f のとき : 斜辺:%f ￥n",
           base, height, hypo);
    return 0;
}

```

6

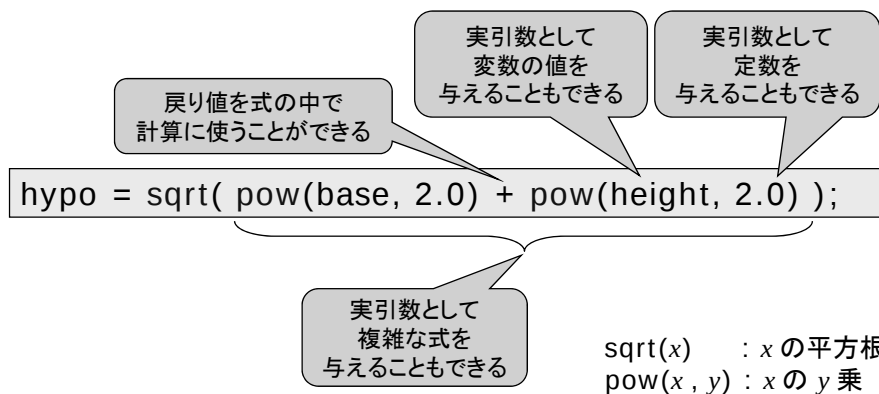
関数の利用法(関数呼び出し)



7

関数呼び出しを含む式

$$\text{height} \quad \text{hypo} \quad \text{base} \Rightarrow \text{hypo} = \sqrt{\text{base}^2 + \text{height}^2}$$



8

関数呼び出しを含む式の計算

base = 3.0 , height = 4.0

```
hypo = sqrt( pow(base, 2.0) + pow(height, 2.0) );
```

```
hypo = sqrt( pow( 3.0, 2.0 ) + pow( 4.0, 2.0 ) );
```

```
hypo = sqrt( 9.0 + 16.0 );
```

```
hypo = sqrt( 25.0 );
```

```
hypo = 5.0
```

9

関数の定義

書式

```
/* 関数の説明 */
戻り値の型 関数名(仮引数の型 仮引数)
{
    return 戻り値を計算する式;
}
```

関数名は自分で命名する
(関数の意味を反映した名前にすること)

実引数の値を
受け取るための変数

仮引数の値を使って戻り値を計算する
「戻り値の型」の式

関数定義の例

```
/* 実引数を2乗した値を求める関数の定義 */
double square(double x)
{
    return x*x;
}
```

10

自作の関数を含むプログラムの構成

書式

```
/* プログラム全体の説明 */
#include <.....>
```

関数のプロトタイプ宣言
(セミコロンを忘れずに!)

```
戻り値の型 関数名(仮引数の型 仮引数); /* 関数の説明 */
```

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

注意:
自分で作成した関数のプロトタイプ宣言を
ファイルの先頭部分(プログラム全体の説明の後)
に記述する。

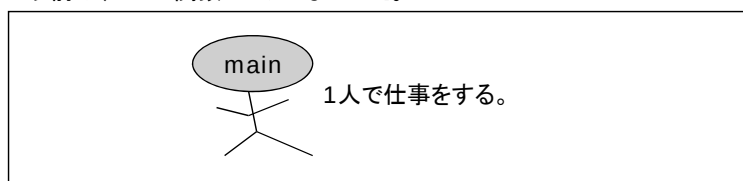
```
/* 関数の説明 */
戻り値の型 関数名(仮引数の型 仮引数)
{
    return 戻り値を計算する式;
}
```

関数の定義
(ここはセミコロン無し)

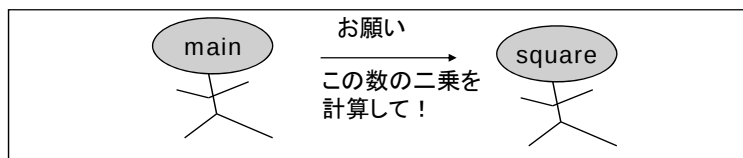
11

イメージ

以前は、main関数1つしかなかった。



main以外の関数定義があると



分業制にできる。
大きなプログラムを書くには、必要な技術。

12

プログラム例2: 自作関数定義の練習

```
/* 関数定義の練習 hypo2.c コメント省略 */
/* 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 */
#include <stdio.h>
#include <math.h>

/* 関数のプロトタイプ宣言 */
double square(double x); /* 実引数を2乗した値を求める関数 */

int main()
{
    double base; /* 直角三角形の底辺の幅 */
    double height; /* 直角三角形の高さ */
    double hypo; /* 直角三角形の斜辺の長さ */

    printf("底辺 ? ￥n");
    scanf("%lf", &base);
    printf("高さ ? ￥n");
    scanf("%lf", &height);

    hypo = sqrt( square(base) + square(height) );

    /* 続く */
```

13

```
/* 続き */

    printf("底辺:%f , 高さ:%f のとき : 斜辺:%f ￥n",
           base, height, hypo);

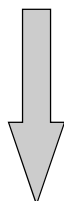
    return 0;
}

/* 実引数を2乗した値を求める関数の定義 */
double square(double x)
{
    return x*x ;
}
```

14

プロトタイプ宣言の役割

プログラムは、
上から下に実行されるので、
プロトタイプ宣言が無いと。



```
int main()
{
    ...
    hypo = ... square(base) ... ;
    ...
    return 0;
}

double square(double x)
{
    return x*x;
}
```

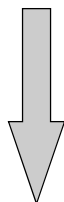
square って
なんだろう？

main

15

プロトタイプ宣言の役割

プロトタイプ宣言があると、
関数であることがわかる。



```
double square(double x) ;

int main()
{
    ...
    hypo = ... square(base) ... ;
    ...
    return 0;
}

double square(double x)
{
    return x*x ;
}
```

square は
実数の値を与えると
実数の値を返すような
関数だ！

main

16

自作関数の呼び出しを含む式

```
double square(double x)
{
    return x*x;
}
```

関数呼び出し時に指定された実引数の値が
仮引数に代入されて、戻り値の計算が行われる。

仮引数の名前は
呼び出し時は気にしない

```
hypo = sqrt( square(base) + square(height) );
```

実引数には変数、定数など
任意の式を与えることができる

17

関数呼び出しを含む式の計算

```
double square(double x)
{
    return x*x;
}
```

base = 3.0 , height = 4.0

```
hypo = sqrt( square(base) + square(height) );
```

```
hypo = sqrt( square( 3.0 ) + square( 4.0 ) );
```

x = 3.0

```
return x * x ;
return 3.0 * 3.0 ;
return 9.0 ;
```

x = 4.0

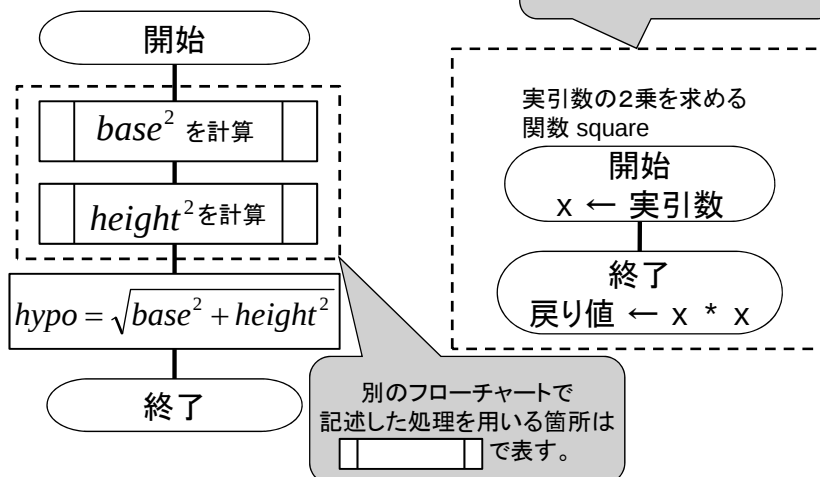
```
return x * x ;
return 4.0 * 4.0 ;
return 16.0 ;
```

```
hypo = sqrt( 9.0 + 16.0 );
```

18

自前の関数を含むプログラムのフローチャート

練習 2 のプログラムのフローチャート
(入出力は省略)



19

より複雑な関数の定義

複数の仮引数を持つ関数

```

戻り値の型 関数名 (仮引数1の型 仮引数1, 仮引数2の型 仮引数2, ...)
{
    return 戻り値を計算する式;
}
    
```

各仮引数ごとに
型を定義

別に定義された自前の関数を利用する関数

```

戻り値の型 関数1(仮引数1の型 仮引数1)
{
    return ...関数2(実引数)...;
}

戻り値の型 関数2(仮引数2の型 仮引数2)
{
    return 関数2の戻り値を計算する式;
}
    
```

外で定義された関数を用いることもできる

20

より複雑な関数の定義の例

関数定義

```
/* 二つの実数値の2乗和を求める関数 */
double sqsum(double a, double b)
{
    return square(a)+square(b);
}

/* 実数値の2乗を求める関数 */
double square(double x)
{
    return x*x;
}
```

使用例

```
hypo=sqsum(base, height);
```

21

プログラム例3: 自作関数を用いた自作関数定義の練習

```
/* 関数定義実験 hypo3.c コメント省略 */
#include <stdio.h>
#include <math.h>

/* 関数のプロトタイプ宣言 */
double square(double x); /* 実引数を2乗した値を求める関数 */
double sqsum(double a, double b); /* 二つの実数値の2乗和を求める関数 */

int main()
{
    double base; /* 直角三角形の底辺の幅 */
    double height; /* 直角三角形の高さ */
    double hypo; /* 直角三角形の斜辺の長さ */

    printf("底辺 ? ￥n");
    scanf("%lf", &base);
    printf("高さ ? ￥n");
    scanf("%lf", &height);

    hypo = sqrt( sqsum(base, height) );
    /* 続く */
```

22

```

/* 続き */

printf("底辺:%f , 高さ:%f のとき : 斜辺:%f \n",
      base, height, hypo);

return 0;
}

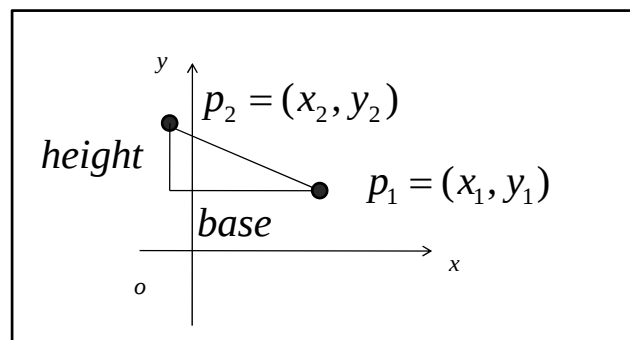
/* 実引数を2乗した値を求める関数の定義 */
double square(double x)
{
    return x*x ;
}

/* 二つの実数値の2乗和を求める関数の定義 */
double sqsum(double a, double b)
{
    return square(a)+square(b) ;
}

```

23

プログラム例4の原理: 平面上で線分の長さを求める計算式



$$base = |x_2 - x_1|, height = |y_2 - y_1|$$

$$hypo = \sqrt{base^2 + height^2}$$

$$L(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

24

プログラム例4: 平面上の線分の長さ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: B0zB0xx
  ソースファイル: lineseg2d.c
  実行ファイル: lineseg2d
  説明: 平面上の線分の長さを求めるプログラム。
        数学関数を利用するため、コンパイルオプション -lm が必要。
  入力: 標準入力から二次元平面上の2つの点(p, qとする)の座標を入力。
        点pのx座標、y座標、点qのx座標、y座標の順に
        4個の実数値(doubleで扱える任意の値)を空白または改行で区切って入力する。
        ただし、点pと点qは等しい座標を持つ点ではないものとする。
  出力: 標準出力に点pと点qを二つの端点とする線分の長さ(正の実数値)を出力。
*/
#include <stdio.h>
#include <math.h>

/* 関数のプロトタイプ宣言 */
double square(double x); /* 実引数を2乗した値を求める関数 */
double sqsum(double a, double b); /* 二つの実数値の2乗和を求める関数 */
double distance(double x1, double y1, double x2, double y2);
/* 点 (x1,y1) と点 (x2,y2) の間の距離を求める関数 */

/* 次に続く */

```

25

```

/* 続き */

int main()
{
  double p_x; /* 一方の端点(点p)のx座標 */
  double p_y; /* 一方の端点(点p)のy座標 */
  double q_x; /* 他方の端点(点q)のx座標 */
  double q_y; /* 他方の端点(点q)のy座標 */
  double length_pq; /* 線分pqの長さ */

  printf("点pの座標?¥n");
  scanf("%lf", &p_x);
  scanf("%lf", &p_y);
  printf("点qの座標?¥n");
  scanf("%lf", &q_x);
  scanf("%lf", &q_y);

  /* 入力値チェック */
  if ( p_x == q_x && p_y == q_y )
  {
    /* 不正な入力のときには、エラー表示してプログラム終了 */
    printf("与えられた二点の座標が等しいため、");
    printf("この二点を端点とする線分は存在しません。¥n");
    return -1;
  }
  /* 正しい入力のとき、これ以降が実行される。 */

  /* 次に続く */

```

26

```

/* 続き */

length_pq = distance(p_x, p_y, q_x, q_y); /* 線分の長さは端点間の距離に等しい */

printf("点p : (%6.2f,%6.2f) %n", p_x, p_y);
printf("点q : (%6.2f,%6.2f) %n", q_x, q_y);
printf("線分pqの長さは%6.2f です。%n", length_pq);

return 0;
}
/* main関数終了 */

/* 実引数を2乗した値を求める関数
   仮引数 x : 2乗すべき値 (任意の実数値)
   戻り値   : xの2乗 (非負の実数値)を返す。
*/
double square(double x)
{
    return x*x ;
}
/* 関数 square の定義終了 */

/* 次に続く */

```

27

```

/* 続き */

/* 二つの実数値の2乗和を求める関数
   仮引数 a : 2乗和を求めるべき値 (任意の実数値)
   仮引数 b : 2乗和を求めるべき値 (任意の実数値)
   戻り値   : aの2乗とbの2乗の和 (非負の実数値)を返す。
*/
double sqsum(double a, double b)
{
    return square(a)+square(b) ;
}
/* 関数 sqsum の定義終了 */

/* 点 (x1,y1) と点 (x2,y2) の間の距離を求める関数
   仮引数 x1 : 一方の点のx座標 (任意の実数値)
   仮引数 y1 : 一方の点のy座標 (任意の実数値)
   仮引数 x2 : 他方の点のx座標 (任意の実数値)
   仮引数 y2 : 他方の点のy座標 (任意の実数値)
   戻り値   : 点 (x1,y1) と点 (x2,y2) の間の距離 (非負の実数値)を返す。
*/
double distance(double x1, double y1, double x2, double y2)
{
    return sqrt(sqsum(x2-x1, y2-y1) ) ;
}
/* 関数 distance の定義終了 */

/* 全てのプログラム(lineseg2d.c)の終了 */

```

プログラム例4の実行結果

```
$make
gcc lineseg2d.c -o lineseg2d

$ ./lineseg2d
点pの座標?
-2 1
点qの座標?
1 5
点p : ( -2.00, 1.00)
点q : ( 1.00, 5.00)
線分pqの長さは 5.00 です。

$
```