

シラバス

7,8. 関数の基礎

関数に関する機能モジュール化方法の理解

事前学修：授業スライド「第4回 関数の基礎」と教科書p. 75～p.77, p.86～p.95, p.108～p.110の熟読。(60分)

事後学修：授業スライドの用語とその意味の理解。関数とはなにか, ライブラリ関数とはなにか, C言語における, 関数の作り方と呼び出し方, 引数と戻り値, return文, void型, プロトタイプ宣言, #include文, 局所変数, を説明できること。(60分)

演習課題のプログラム及びレポートの完成と提出。(180分)



プログラミングの基礎及び演習

情報工学科



プログラミングの基礎 第4回

■ 関数

- 関数とは何か？
- 自作の関数を作るには？

- 関数の定義
- 引数と戻り値
- 関数呼び出し
- プロトタイプ宣言

- 関数と変数
- ライブラリ関数

- 教科書p.75～p.77, p.86～p.95, p.108～p.110
-

関数とは何か？

- 関数：あるまとまった仕事（処理）をするもの
 - 何度も行う処理を自作の関数にして用いる
 - C言語に組み込まれていない機能を追加するためにライブラリ関数を用いる

例：

- printf
 - － 「画面表示」という仕事をする関数
- scanf
 - － 「キーボードから変数へのデータ入力」という仕事をする関数

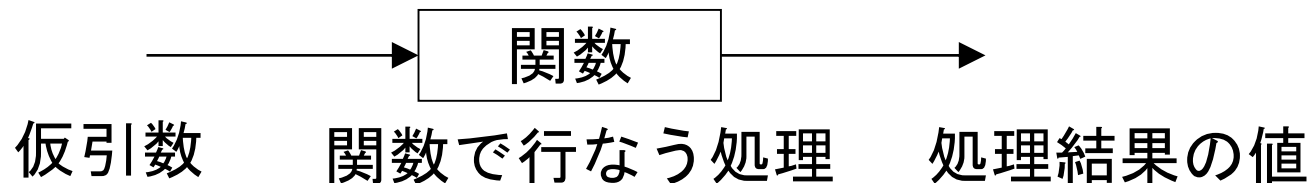
- C言語では、関数を作り、それらを組み合わせることでプログラムを作成する
-

自分で関数を作るには？

- 関数は自分で作ることができる
 - C言語のプログラムを作るというのは、関数を作るということ
- 関数を作る
 - 作成する関数のプログラム中に、必要な処理を記述する
- その関数を呼び出す
 - mainの中などから、自分で作成した関数を呼び出して使用する

なぜ関数を作るのか？

- 同じ処理を複数回書かないようにするため
- 「処理のまとめり」を関数にすると、
誤りが少なくなる
 - 「構造化プログラミング」という
 - 「モジュール化」ともいう
 - main関数に長々と書くと、
誤りが入りやすいことが経験的に知られている



関数の作り方

```
戻り値の型 関数名( 型 仮引数1, 型 仮引数2, ... ) {  
  <局所変数の宣言>  
  <処理の中身（ボディ）>  
  return (戻り値);  
}
```

■ 戻り値の型

- 関数の実行が終了したときに出力として返す戻り値の型

■ 関数名

■ 引数ならび（型 仮引数1, 型 仮引数2, ...）

- 関数に与えるデータ（型と名前）,
引数がない場合もある

■ return (戻り値)

- 関数が返す戻り値があるときに, returnで戻す.

関数の例 (1)

```
#include <stdio.h>
int keisan(int m, int n) {
    int a;
    a = m + n;
    return a;
}
```

(2) 5がmに, 3がnにコピーされ,
m+nの計算結果がaに格納

(3) 計算結果a=8を呼び出し元に戻す

```
int main(void) {
    int x;
    x = keisan(5, 3);
    printf("x = %d\n", x);
    return 0;
}
```

(4) xに値8が格納される

(1) “5+3の計算をしてくれ”
と関数keisanを呼び出す



関数の例 (2)

```
#include <stdio.h>
int keisan(int m, int n) {
    int a;
    a = m + n;
    return a;
}

int main(void) {
    int x;
    x = keisan(5, 3);
    printf("x = %d\n", x);
    return 0;
}
```

keisan関数の定義

main関数の他に、
自分で作成した関数
keisanを記述する。
keisanだけ、あるいは
mainだけでは動かない。

keisan関数を呼び出す



関数の例 (3)

型宣言

関数を実行すると、
呼び出し元にint型の戻り値を返す

関数の名前

keisanという名前

引数ならび

int型の引数を2つ受け取る
keisanの内部では、
この2つの引数をm, nという名前の
変数として扱う

ボディ

2つの引数の値を加算する

return文

呼び出し元に変数aの戻り値を返す

```
int keisan( int m, int n ) {  
    int a;  
    a = m + n;  
    return a;  
}
```

p.88, 例題7

プログラミングの基礎 第4回

■ 関数

- 関数とは何か？
 - 自作の関数を作るには？
 - 関数の定義
 - 引数と戻り値
 - 関数呼び出し
 - プロトタイプ宣言
 - 関数と変数
 - ライブラリ関数
-
- 教科書p.75～p.77, p.86～p.95, p.108～p.110

引数と戻り値： return文

- ある関数の中から、
呼び出し元に戻り値を返す
 - 関数の戻り値を返さない場合、
return文は書かなくてもよい。
- 例：
 - mainから関数keisanを呼び出す
 - keisanの中で計算処理を行なう
 - 計算結果をkeisanからmainに戻したい
 - returnを使うとkeisanからmainに戻り値が返る

引数と戻り値：void型

- 戻り値を返さない
 - 型宣言にvoidと書く
 - 例：画面に表示するだけ
- 引数を受け取らない
 - 引数ならびに void と書く

```
double func1(double x, double y) {  
    ...  
    return z;  
}
```

double型の戻り値zを返す
引数はdouble型のxとy

等

```
void func2(void) {  
    ...  
}
```

戻り値を返さない
引数もない

```
void func3(int x) {  
    ...  
}
```

戻り値を返さない
引数はint型のx

関数の呼び出し

■ 関数の呼び出し方法 (p.90)

関数名(引数ならび)

例 1 :

関数func1を呼び出す.
引数a, bを渡す.
func1から返ってきた
戻り値は変数yに
代入される.

```
double func1(double x, double y) {  
    ...  
    return z;  
}  
  
int main(void) {  
    double a, b, y;  
    . . . . .  
    y = func1( a, b );  
    . . . . .  
}
```

関数の呼び出し

■ 関数の呼び出し方法 (p.90)

関数名(引数ならび)

例 2 :

関数func2を呼び出す.
引数はない.
戻り値も返ってこない.

```
void func2(void) {  
    ...  
}  
  
int main(void) {  
    ....  
    func2();  
    ....  
}
```

プログラミングの基礎 第4回

■ 関数

- 関数とは何か？
 - 自作の関数を作るには？
 - 関数の定義
 - 引数と戻り値
 - 関数呼び出し
 - プロトタイプ宣言
 - 関数と変数
 - ライブラリ関数
-
- 教科書p.75～p.77, p.86～p.95, p.108～p.110

プロトタイプ宣言 (1)

■ プロトタイプ宣言

- 関数の形式をプログラムの先頭で宣言すること
- **全ての関数**について, プロトタイプ宣言が必要

■ 書式

- **関数定義の1行目と同じものに**,
セミコロンを付けて,
プログラムの**先頭部分**に記述

型 関数名 (型 仮引数1, 型 仮引数2...);

プロトタイプ宣言 (2)

■ なぜ，プロトタイプ宣言が必要なのか？

□ 関数の定義より前にその関数を使う場合

- プロトタイプ宣言がない場合，
コンパイルエラーとなる

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

```
int func1( int a, int b );
```

```
void func2( void );
```

```
int main( void ) {  
    int a=2, b = 5, y;  
    y = func1( a, b );  
    printf( "y = %d¥n", y );  
    func2();  
    return 0;  
}
```

```
int func1( int a, int b ) {
```

```
    int c;  
    c = a + b;  
    return c;  
}
```

```
void func2( void ) {  
    printf( "end of computing¥n" );  
}
```

main関数

■ mainもC言語の関数

- C言語のプログラムを実行したときに、最初に呼び出される関数
- main関数のみプロトタイプ宣言は不要

■ int main(void)

- main関数はint型の戻り値を返す関数
- 引数はなし (void)

■ 戻り値には任意の値を返せるが、一般的に

- 成功の場合は、0を利用
- 失敗の場合は、0以外の値を利用

プログラミングの基礎 第4回

■ 関数

- 関数とは何か？
- 自作の関数を作るには？

- 関数の定義
- 引数と戻り値
- 関数呼び出し
- プロトタイプ宣言

□ 関数と変数

□ ライブラリ関数

- 教科書p.75～p.77, p.86～p.95, p.108～p.110
-

関数と変数 (1)

■ 局所変数

□ 関数の中だけで有効な変数

```
#include <stdio.h>
int keisan( int x, int y );
int main( void ) {
    int a, b, c;
    a = 2;
    b = 3;
    c = keisan( a, b );
    printf( "sum = %d\n", c );
}
int keisan( int x, int y ) {
    int z;
    z = x + y;
    return z;
}
```

a,b,cはmain関数の中でのみ使用可能
a,b,cをkeisan関数の中で使うとエラー

x,y,zはkeisan関数の中でのみ使用可能
x,y,zをmain関数の中で使うとエラー

関数の中で計算に必要なデータは、
すべて引数として渡す必要がある
main関数のa,bを両方とも
keisanに渡さないと計算できない

関数と変数 (2)

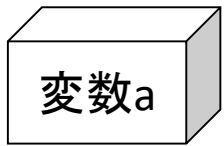
■ 局所変数は、各関数ごとに独立

□ 別の関数にある同じ名前の局所変数は互いに影響しない

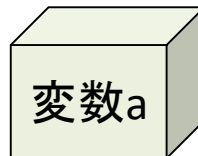
- main関数、関数func1と関数func2で宣言している変数aは異なる変数。同様に変数bも異なる。

```
int main(void) {  
    int a, b, ...  
    . . . .  
}
```

func1の変数a



func2の変数a



同じ名前だが、別の“箱”

```
int func1( int a ) {  
    int b;  
    b = a + 5;  
    return b;  
}
```

```
int func2( int a ) {  
    int b;  
    b = 5*a;  
    return b;  
}
```

関数と変数 (3)

■ 実引数

- 値を関数に渡すために使われる変数

■ 仮引数

- 関数側で値を受け取るための変数

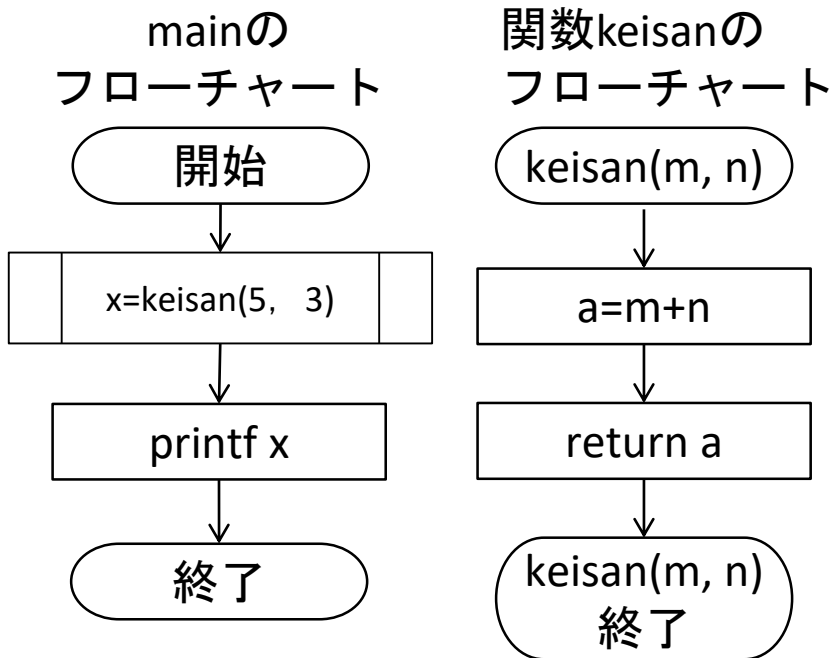
```
#include <stdio.h>
int keisan( int x, int y );
int main( void ) {
    int a, b, c;
    a = 2;
    b = 3;
    c = keisan( a, b );
    printf( "sum = %d n", c );
}
int keisan( int x, int y ) {
    int z;
    z = x + y;
    return z;
}
```

keisanに値を渡しているa,bは
keisanの実引数.

keisanで値を受け取るx,yは
keisanの仮引数.

関数のフローチャート

関数毎にフローチャートを作成



```
#include <stdio.h>
int keisan(int m, int n) {
    int a;
    a = m + n;
    return a;
}

int main(void) {
    int x;
    x = keisan(5, 3);
    printf("x = %d\n", x);
    return 0;
}
```

関数の変数表

■ 関数毎に変数表を作成する

関数func1の変数表

変数名	型	内容	条件
score	int	点数	$100 \geq x \geq 0$

関数func2の変数表

変数名	型	内容	条件
temperature	double	気温	$100 \geq x \geq -273$



プログラミングの基礎 第4回

■ 関数

- 関数とは何か？
 - 自作の関数を作るには？
 - 関数の定義
 - 引数と戻り値
 - 関数呼び出し
 - プロトタイプ宣言
 - 関数と変数
 - ライブラリ関数
-
- 教科書p.75～p.77, p.86～p.95, p.108～p.110

ライブラリ関数

- C言語には、あらかじめ用意されている多数の関数があり、**ライブラリ関数**と呼ぶ
 - 例： `printf`, `scanf`など
 - ライブラリ関数は自由に使える
 - 行ないたい処理がライブラリ関数として用意されていれば、それを使えば良い.
 - 教科書p.205以降に
主要なライブラリ関数がまとめられている

数学ライブラリ関数

■ 数学ライブラリ関数

□ ライブラリ関数の一例

□ 三角関数, 平方根, 対数など

関数名	数学での書き方	C言語での書き方
sin	$\sin x$	<code>sin(x)</code>
cos	$\cos x$	<code>cos(x)</code>
tan	$\tan x$	<code>tan(x)</code>
sqrt	$\sqrt{x}$	<code>sqrt(x)</code>

関数名	数学での書き方	C言語での書き方
exp	e^x	<code>exp(x)</code>
log	$\log_e x$	<code>log(x)</code>
log10	$\log_{10} x$	<code>log10(x)</code>
pow	x^y	<code>pow(x, y)</code>

変数 x, y はdouble型

※三角関数 $\sin(x)$ などの引数 x はラジアンで与える

数学関数の使い方

```
#include <stdio.h>
#include <math.h>
int main(void) {
    double x, y;
    printf( "x = " );
    scanf( "%lf", &x );
    y = sin( x );
    printf( "sin(x) = %f¥n", y );
    return 0;
}
```

数学関数を使うときに必要

数学関数には実数値を使う

数学関数を呼び出す
sin(x)を計算し、
結果を変数yに代入する

数学関数の補足

■ 数学関数を使うときのコンパイル方法

□ (gccを使う場合のみ) 教科書p.77

□ gcc ソースファイル名 **-lm**
エル・エム

例：

```
gcc reidai6-1.c -lm
```

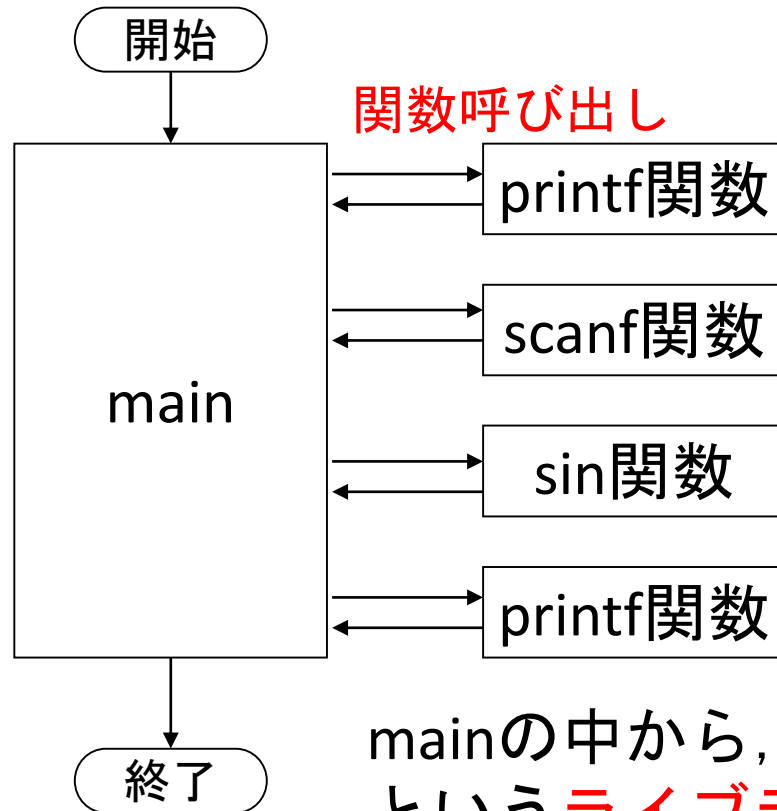
-lmオプションを付けないとリンクエラーが発生するので注意

やや高度な説明：

-lmは、数学ライブラリをリンクするためのオプション



例題6.1の解説 (1)



```
#include <stdio.h>
#include <math.h>
int main(void) {
    double x, y;
    printf( "x = " );
    scanf( "%lf", &x );
    y = sin( x );
    printf( "sin(x) = %f¥n", y );
    return 0;
}
```

mainの中から, printf, scanf, sin
という**ライブラリ関数**を呼び出す。
ライブラリ関数の処理が終了したら,
mainに処理が戻る。



#include <stdio.h> (1)

- プロトタイプ宣言は全ての関数に必要
 - printf, scanfなどのプロトタイプ宣言は？

- ライブラリ関数のプロトタイプ宣言は、別のファイルの中に記述されている。
 - #include命令で、そのファイルを取り込む
 - printf, scanf
stdio.h というファイル内でプロトタイプ宣言
 - sin, cos, exp
math.h というファイル内でプロトタイプ宣言
 - stdio.hやmath.hは
システム上に事前に用意されている

※ライブラリ関数は後述



#include <stdio.h> (2)

■ #include <stdio.h>

- printf, scanfのプロトタイプ宣言が行われる
 - ・ ソースファイルの中の#include <stdio.h>の部分に, printf, scanfなどのプロトタイプ宣言がコンパイラによって追加される

■ manコマンドを使うことでどのファイルを#includeする必要があるか分かる

- 例 : man printf

■ 参考: #include <math.h>

- 数学関数はmath.h内でプロトタイプ宣言されている
-

注意：関数ではないもの

■ 制御文は関数ではない

- if, switch, for, while, breakなどは関数ではない
- 制御文：プログラムの処理の流れを制御する
- 制御文は，C言語自身が持っている機能
 - ・ 関数：C言語自身が持っていない機能を外部から追加するもの
- 正） if文， main関数
- 誤） if関数， main文

■ 変数宣言文は関数ではない

- int, doubleなどの変数宣言も関数ではない
-

【演習課題】

- 演習課題提出システムの「第4回」演習課題を実施
 - 演習課題4-1～4-6
 - 演習時間に，設計後に演習環境にリモートログインしプログラムを作成
 - 演習環境でコンパイル，テストし，完成
 - ソースファイル（4-x.c）をWinSCP等でPCへ転送
 - 課題提出システムへSubmitし各課題100点をを目指す
 - 時間内に終了しない場合 ⇒ 宿題
 - 授業翌週水曜日の午後5時までに必ず提出
-

【レポート】

- 対象：演習課題4-3～4-6
- 授業資料とともにアップした”レポートファイル”に下記を作成
 - テスト仕様書を”レポートファイル”に貼り付け.
 - 変数表を”レポートファイル”に貼り付け.
 - フローチャート（astah*にて作成し「ツール」, 「画像出力」にてPNG形式ファイルを作成し”レポートファイル”に貼り付け）
 - ソースコード, コンパイル結果, 実行結果をscriptコマンドでファイルにし印刷し”レポートファイル”に貼り付け.
- 授業翌週水曜日の午後5時までに
55号館304室前のレポート提出小箱へ提出

ヒント：課題4-2

偶数が入力されない時に、関数が呼ばれて
関数内でエラーメッセージを表示する

実行例1

Input an even number : 4
The input value 4 is even.

..... 関数error_messageは呼ばれない

実行例2

Input an even number : 5
ERROR : invalid value!

..... 関数error_message内で表示



ヒント：課題4-3

月の入力部分で、関数get_monthが呼ばれる

実行例1

What month is it now?

MONTH : 11

This month is November.

赤枠内がget_month関数内での処理

実行例2

What month is it now?

MONTH : 0

ERROR : invalid value!

MONTH : 2

This month is February.



ヒント：課題4-4

アスタリスクの表示だけを関数draw_charで行う

実行例1

number of asterisks : 3

実行例2

number of asterisks : 0

赤枠内がdraw_char関数内での処理

実行例3

number of asterisks : -2

ERROR : invalid value!

number of asterisks : 5

ヒント：課題4-5

2点間の距離の計算を関数distanceで行う

実行例

<< Point A >>

x coordinate value : 1 return

y coordinate value : 2 return

<< Point B >>

x coordinate value : 3 return

y coordinate value : 4 return

The distance between two points is 2.828427.

プログラムの流れでは、この位置で
関数distanceが呼び出されている

ヒント：課題4-6

月の入力部分で、関数get_monthが呼ばれる
get_month関数内で、月の入力値が間違っている場合、
関数error_messageが呼ばれてエラーメッセージを表示する

実行例1

What month were you born?

MONTH : 2

What month is it now?

MONTH : 11

There are 3 months until your birth month.

赤枠内がget_month関数内での処理

緑枠内がerror_message関数内での
処理

実行例2

What month were you born?

MONTH : 0

ERROR : invalid value!

MONTH : 2

What month is it now?

MONTH : 22

ERROR : invalid value!

MONTH : 12

There are 2 months until your birth month.