

C プログラミング 入門

— 関数・グローバル変数 —

早稲田大学

本日の目標

- 関数が作れる
 - 書き方
 - 呼び出し
 - ローカル変数
 - 変数のスコープ（有効範囲）と持続時間
- グローバル変数を理解する
 - 使い方
 - スコープと持続時間
 - 配列の場合

```
#include <stdio.h>
戻り値の型 関数名 ( 引数の型 引数名 ) {
    ...
    return 戻り値;          /*この関数の戻り値を return 文で指定*/
}
int main(void){
    double a;
    a = 関数名 ( 引数 );    /*上の関数を呼び出し、戻り値を a に代入*/
}
```

- 関数には、**引数**と**戻り値**がある
- 通常、`#include` 文の後、`main` 関数の前に書く
(関数が見えるのはソースファイル中のその場所以降)
- 関数名のルールは変数名と同じ、役割が分かりやすい名前をつける
- **return** 文で関数を終了する。**戻り値**はこの時に指定する
- 引数は複数指定できるが、戻り値は1つのみ
- 関数を使うときは、情報を**引数**として渡して呼び出し、**戻り値**を受け取る

係数固定の1次関数を作る

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

```
int main(void){  
    double a[3];  
    a[0]=2.5*0+1.0;  
    a[1]=2.5*1+1.0;  
    a[2]=2.5*2+1.0;  
    return 0;  
}
```

```
#include <stdio.h>  
double linear(int x){  
    double y;  
    y = 2.5*x+1.0  
    return y;  
    /*呼び出し元に返したい値*/  
}  
int main(void){  
    double a[3];  
    a[0]=linear(0);  
    a[1]=linear(1);  
    a[2]=linear(2);  
    return 0;  
}
```

- 共通した部分「 $2.5*n+1.0$ 」を何度も書かなくてもすむ

係数固定の1次関数を作る

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

```
int main(void){  
    double a[3];  
    a[0]=2.5*0+1.0;  
    a[1]=2.5*1+1.0;  
    a[2]=2.5*2+1.0;  
    return 0;  
}
```

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

```
double linear(int x){  
    double y;  
    y = 2.5*x+1.0  
    return y;  
    /*呼び出し元に返したい値*/  
}  
int main(void){  
    double a[3];  
    a[0]=linear(0);  
    a[1]=linear(1);  
    a[2]=linear(2);  
    return 0;  
}
```

- 共通した部分「 $2.5*n+1.0$ 」を何度も書かなくてもすむ

少し詳しい話

- C 言語のプログラムは関数の寄せ集めで構成される
 - 実は `int main(void){...}` も関数の 1 つ
 - プログラム内には必ず `main` 関数が必要
- 関数の種類
 - **main 関数** (プログラムの実行 $\approx$ `main` 関数の実行)
 - 標準で用意されている関数 (`#include` により使用できる)
 - 自分で名前がつけられる
 - **戻り値**を求めるもの (例: $\sin(x)$ や $\log(x)$ など)
 - **戻り値**が無く, 関数内での操作が主目的なもの (例: `printf("Hello, world\n");` など)
- 関数を作ると便利になること
 - プログラムの共通する部分を一箇所にまとめることができる
 - プログラムを機能的に分かりやすい単位で書ける

例題

例題：実数 x を入力したとき次の最大値を求めるプログラムを作れ

- 実数 x を入力すると, x , $-x$, x^2 , $\sqrt{|x|}$ の中で一番大きい値を答えるプログラムを作れ (ファイル名は `dmax.c` とする) .
- 表示は以下のようにする .

Input x: -0.5 【Enter】

Answer is 0.707107.

例題のヒント

- $\max(a, b)$ という関数を作り, a と b の内小さい方を返すようにする. この関数を main 関数内で呼び出す

```
double max(double a, double b){  
    if (a<b) return ... ;  
    else     return ... ;  
}
```

- 実数の絶対値は「fabs()」, ルートは「sqrt()」という関数が数学ライブラリにある
- 数学ライブラリを使うには,
(1) #include <math.h>, (2) -lm オプション
- 仮の最大値を記憶させておくために, 新たな変数を用意することに気づくこと (変数名は何でも良い)

例題のヒント

メインの部分は以下のようにする

関数名：main

```
int main(void){  
    double x, y;  
    printf("Input x:");  
    scanf("%lf",&x);  
    y = max(x, -x);  
    y = max(y, x*x);  
    ...  
    printf("Answer is %f.¥n",y);  
    return 0;  
}
```

- x には入力値, y には仮の最大値を入れる
- 関数を何回か呼び出して戻り値を y に代入

引数と変数

- 関数内で引数の値を変更しても，呼び出した引数の値は**変更されない**
- 別の関数内なら，同じ名前の変数を宣言しても**別物**

```
#include <stdio.h>
void func(int x){          /* 引数 x はこの関数内だけ */
    x=7;                   /* x を変更しても下の x は影響を受けない */
}
int main(void){
    int x=3;
    func(x);               /* x=3 が上の x にコピーされる */
    printf("x is %d.\n", x); /* 「x is 3.」 */
    return 0;
}
```

変数のスコープ（有効範囲）

- すべての変数は，ソースファイルのどの部分から使用（読み書き）できるか（スコープ）が決められている
- 関数内で宣言した変数（仮引数も含める）はローカル変数と呼ばれ，宣言された関数内だけで使用可能
- 関数外で宣言した変数はグローバル変数と呼ばれ，すべての関数で使用可能
- 別の関数内で同じ名前の変数を宣言して使用することも可能．これらは別物として扱われる

```
double functionA (int x, int y){  
    ...  
}  
int functionB(int x){  
    double y;  
    ...  
}
```

/* 上の x とこの x は別物*/
/* 上の y とこの y は別物*/

変数の持続時間

- 変数は，プログラム実行時に，いつ使用可能になり，いつ使用不可能になるか（**持続時間**）が決められている
- 関数内で宣言した変数（**引数**も含める）は**ローカル変数**と呼ばれ，宣言した関数が呼び出されてから `return` で戻るまでの間だけ，記憶領域（メモリ）が確保され，値を保持しておく．使用できるのはこの間だけ（`main` で使った変数と同じ名前が使える）
- **グローバル変数**の場合，プログラムの実行が始まってから終了するまでずっと使用できる
- 再帰呼び出しなどで呼び出しが重なった場合は，呼び出しの都度新たに別物の変数が用意される（参考）

グローバル変数

⇒ プログラムのどの部分からもアクセス可能な変数

- 関数外で宣言する（通常は#include 文や #define 文の後に書き，その後に関数が続く．宣言以降で使用可能）
- **スコープ**：宣言以降ソースファイルの最後まで
- **持続時間**：プログラムの実行が始まってから終了するまで

```
#include <stdio.h>
double ParamA, ParamB; /* 1. グローバル変数宣言*/

double linear(double x){
    return ParamA*x+ParamB; /* 3. 変数を使う*/
}

int main(void){
    ParamA=5.0; ParamB=3.2; /* 2. 値を設定*/
    printf("Result is %f.¥n", linear(1.2));
    printf("Result is %f.¥n", linear(2.0));
    return 0;
}
```

グローバル変数（参考）

巨大な配列は，ローカル変数にできない（ローカル変数用に使えるメモリ量が少ない）

例

```
int main(void){
    double Array[2000000000];
    /*ローカルで巨大配列は使えない．グローバルで宣言する*/
    ...
}
```

- グローバル変数は明示的に初期化しない場合は 0 に初期化される．（ローカル変数を初期化しないと値は不定）
- 巨大配列の部分初期化は非効率なので注意（実行ファイルのサイズが巨大になる等）

```
double Array[2000000000] = {1.0, 2.0};
```

関数の宣言いろいろ

```
int method1(int a, int b)
```

- 2つの int 型の引数を受け取り,
int 型の戻り値を返す関数

```
double method2(int a, double b)
```

- int 型と double 型の引数を1つずつ受け取り,
double 型の戻り値を返す関数

```
boolean method3(int a, int b, int c)
```

- 3つの int 型の引数を受け取り,
boolean 型 (true または false) の戻り値を返す関数

```
void method4(double x)
```

- double 型の引数を1つ受け取り, 戻り値のない関数

```
double method5(void)
```

- 引数を受け取らず, double 型の戻り値を返す関数

まとめ

- 関数が作れる
 - 書き方
 - 呼び出し
 - ローカル変数
 - 変数のスコープ（有効範囲）と持続時間
- グローバル変数を理解する
 - 使い方
 - スコープと持続時間
 - 配列の場合