# プログラミング演習
# ～関数～

## 1　目的

C 言語における関数の使い方を理解する．またバンディット問題 [1] 等を例題にしながら実践し理解を深める．

## 2　製作対象 関数の基礎

関数とは決まりきった処理やよく使う処理をまとめたものである．これによって，変更・修正や使用の手間を簡便化できる．一般的には，関数は”入力”を与えられると，入力に応じた計算・演算を行い，計算結果・演算結果を”出力する”．その関数の基本的な定義は

- 出力の型　関数の名前 (入力の型および変数名, 入力の型および変数名, ...)
  - 例: int f(float a, double b);

となる．このように，複数の変数を入力することができ，計算結果・演算結果は 1 種類 1 個の値となる (上記例では出力は int 型の数値となる)．

関数の基本的な使い方は，2.2 プログラムとなる．

まず，コンパイラに自分が作った関数について教えないといけない．これが 7 行目で，関数を使うための最低限必要な情報 (関数の名前・入出力の型と個数) を記述している．これを関数の定義という．

次に，関数を自分で作る必要がある．これが 24 行目から 30 行目となる．ここでは入力として x,y があり，この和を計算して出力する関数を例とした．計算結果は，29 行目の return 変数 (または値); で出力することとなる．これを関数の実体という．

そして自分で作った関数を使っている部分が，18 行目である．このように，x,y の値を入力することで関数 sum は和を計算し，return で出力した値が z に代入されている．これを関数の使用という．

このように，関数を使うことで，同じ処理 (この場合には和) を 1 行で，どこでも，何箇所でも使用できる．そして，間違いの修正や計算・演算の処理を変更したい場合には，24 行目以降の関数の中身を修正・変更することで，複数箇所で関数を使っていても一ヶ所の修正・変更ですべての箇所で修正・変更したことになる．

## 2.1 準備

ディレクトリ ”programming09”を作成する．今回の演習では，プログラムの作成や必要ファイルのダウンロードは，programming09 ディレクトリで行う．

## 2.2 プログラム

```
1  /*********************************
2        関数の基本的な使い方
3  *********************************/
4
5  #include <stdio.h>
6
7  int sum(int x, int y);
8
9  int main(){
10   int x,y,z;
11
12   printf("一つ目の数値を入力してください:");
13   scanf("%d", &x);
14
15   printf("二つ目の数値を入力してください:");
16   scanf("%d", &y);
17
18   z=sum(x,y);
19
20   printf("和は%d です\n",z);
21   return 0;
22 }
23
24 int sum(int x, int y){
25   /* 関数の処理 */
26   int a;
27
28   a=x+y;
29   return a;
30 }
```

## 2.3 コンパイル

上記のプログラムを作成し，ファイル名 func-example.c として保存してコンパイルを行う．コンパイル後の名前を exfunc とする．

> gcc -o exfunc func-example.c

## 2.4 動作実験

コンパイルしたプログラムを実行し，和計算が行われていることを確認する．

## 2.5 調べてみよう

関数の詳細な使い方について教科書等で調べてみよう．

## 2.6 やってみよう

### 2.6.1 差乗除関数の作成

2.2 プログラムでは二つの整数を入力とし，和を計算し出力する関数を作成した．同じように，差 (引き算) を計算し出力する関数，乗 (掛け算) を計算し出力する関数，除 (割り算) を計算し出力する関数を作ってみよう．特に，除算では 0 で割ると問題が生じるので気をつけること．

# 3 製作対象 作った関数の整理-ファイルの分割とヘッダファイル-

一つの関数しか作成しなければ整理は必要ないが，数が多くなると全体が見えにくくなり，整理が必要となってくる．ここではファイルを分割して整理する方法について考える．

ここでこれまでのプログラムでよく見かけた (はずの)#include に注目する．これは他のファイルを読み込む命令である (図 1)．これより，2.2 プログラムで作成した和の関数の定義と実体を新しいファイル my-calc に移し，#include で読み込むことにより，2.2 プログラムと同等なものを作ることができる (図 2 参照．実感してみたければやってみよう)．ちなみに，#include<stdio.h> のように < と > でファイルをくくるのは標準ファイル用であり，”と”のダブルクオーテーションでファイルをくくるのはユーザが作成したファイルである．特に挙動に大きな違いはない (細かい違いはある可能性がある)．

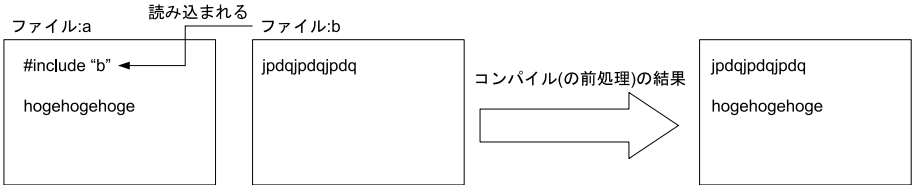


図 1: include 命令による他ファイルの取り込み

和だけでなく差乗除の関数も作っている場合，ファイル:my-cal に差乗除の関数の定義と実体も移動すると，ファイル:my-calc は自分で作った四則演算の関数群となり，他のプログラムでも使用できるようになる．

ここで，更に定義と実体を二つのファイルに分け，整理することを考える．定義ファイル (これをヘッダファイルと呼ぶ) は一般に拡張子.h をつけ，実体ファイルは一般に拡張子.c(C 言語の場合) をつける．この二つのファイルを include することで，2.2 プログラムと同等なものを作ることができる (図 3 参照．実感してみたければやってみよう)．

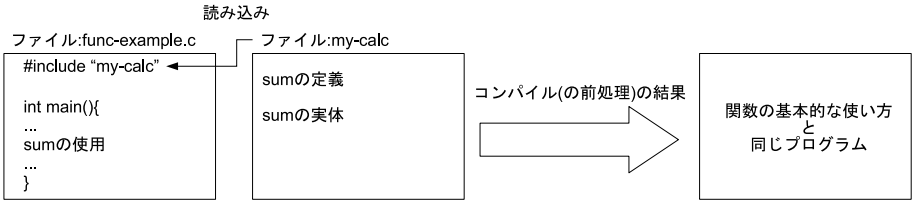


図 2: include 命令による自作関数の取り込み

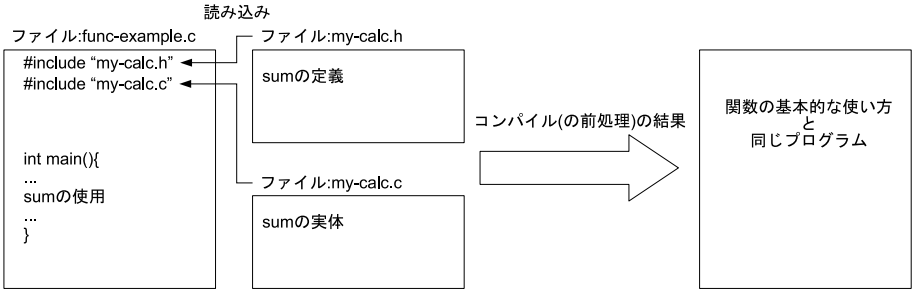


図 3: include 命令によるの定義・実体ファイルの取り込み

これまで述べていた”コンパイル”は，実行ファイルを作成する行為であったが，厳密には間違いである．実行ファイルを作成する行為は，コンパイル (機械語への翻訳) とリンク (複数ファイルの結合) に分けられ，この二つを実行して始めて実行ファイルが作られる．

ここで，コンパイルとはユーザが作成したプログラムを機械語へ翻訳することであり，その結果オブジェクトファイル (拡張子.o) を作る．このオブジェクトファイルのみではプログラムは実行できない．なぜならば，printf や scanf などシステムが用意している他のオブジェクトファイルが必要となるためである．そこで，必要なオブジェクトファイル群を集め，結合することで実行ファイルを作成する．これがリンクである．実行ファイルを作る行為をコンパイル (広義)，機械語への翻訳 (オブジェクトファイルの作成) のみをコンパイル (狭義) とすると

- コンパイル (広義)=コンパイル (狭義)+リンク

である (図 4).

図 3 に示す定義ファイル (ヘッダファイル) は読み込まなければならないが，実は，実体ファイルは#include で読み込む必要はない．下記のコマンドで，自動的に my-calc.c もオブジェクトに変換し，自動的にリンクしてくれるためである．

> gcc -o func-example func-example.c my-calc.c

ここで，my-calc.c をコンパイルのみしてオブジェクトファイルを作る場合には，

> gcc -c my-calc.c

と実行すると，オブジェクトファイル my-calc.o を作ることができる．作成したオブジェクトファイルを用いて

> gcc -o func-example func-example.c my-calc.o

とすることもできる (この場合，my-calc.c を再びコンパイル (狭義) せず，存在するオブジェクトファイルを使用する).

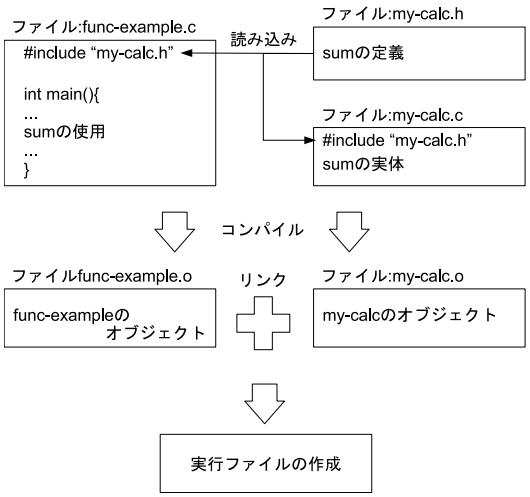


図 4: コンパイル (広義) の概要

### 3.1 プログラム

2.2 で作ったプログラムを func-example.c と my-calc.h，my-calc.c に分割し，分割コンパイルを実現する．

### 3.2 コンパイル

下記コマンドによって，オブジェクトファイルの作成を確認する．

> gcc -c my-calc.c

下記コマンドによって，実行ファイル func-example を作成する．

> gcc -o func-example func-example.c my-calc.o

下記コマンドによって，実行ファイル func-example を作成する．通常の簡易プログラムでは，これが一般的な方法である．

> gcc -o func-example func-example.c my-calc.c

### 3.3 動作実験

作成したプログラムを実行し，分割コンパイルが出来ていることを確認する．

このように，関数の入出力および名前さえ明確に決めて変更しなければ，関数の実体の変更・修正・バージョンアップの影響は関係するファイルの中だけとなる．

## 4 製作対象 バンディットプログラム -腕の 3 試行一括選択-

配列の演習 [2] 時に行った複数回の腕の選択を一括に行う関数の作成を考える．今回は 3 試行分 (3 回分) の腕の選択を一度に行うことを考える．その為に，関数の定義 (入出力・名前) を 4.2 プログラムの 9 行目のように考える．実体は 35 行目から，そして使用は 28 行目となる．

## 4.1 準備

はたおり虫より，以下のファイルをダウンロードする．

- bandit.h
- bandit00.o

## 4.2 プログラム

バンディット関係の関数は，ガイダンス資料 [1] に説明があるので一読すること．

```
1  /*********************************************************
2                    腕の一括選択 (3 試行)
3  *********************************************************/
4
5  #include <stdio.h>  /* システムの用意した入出力        */
6  #include <stdlib.h> /* システムの用意した便利関数      */
7  #include "bandit.h" /* ユーザが用意したバンデット関連 */
8
9  double armselect3(int first, int second, int third);
10
11 int main(){
12   int num_arm, loop, arm_select, selection[3];
13   double total_reward;
14
15   init_bandit();           /* バンディットの初期化 */
16
17   num_arm = get_arm_num();/* バンディットの腕の数を取得 */
18   printf("このバンディットの腕の数は%d です\n",num_arm);
19
20   /* バンディットの腕の選択 (3 試行分) */
21   for(loop=0 ; loop<3 ; loop++){
22     printf("バンデットの腕の番号を選択してください [1-%d]:",num_arm);
23     scanf("%d", &arm_select);
24     selection[loop] = arm_select;
25   }
26
27   /* 関数 armselect3 による一括選択 */
28   total_reward = armselect3(selection[0], selection[1], selection[2]);
29
30   printf("総スコア:%lf\n",total_reward);
31
32   return 0;
33 }
```

```
34
35  double armselect3(int first, int second, int third){
36    int loop;
37    double total;
38
39    /* バンディット動作 3回 */
40    total = bandit(first);
41    total = total + bandit(second);
42    total = total + bandit(third);
43
44    return total;
45  }
```

### 4.3 コンパイル

上記のプログラムを作成し，ファイル名 gameplay-armselect3.c として保存してコンパイルを行う．コンパイル後の名前を gameplay-armselect3 とする．

> gcc -o gameplay-armselect3 gameplay-armselect3.c bandit00.o

### 4.4 動作実験

作成したプログラムを動作させ，3 回腕が選択されていることを確認する．

### 4.5 やってみよう

#### 4.5.1 ファイル分割

ファイル分割し，armselect3 関数を armselect3.h と armselect3.c に記述し，実行ファイルを作成してみよう．

#### 4.5.2 選択できる腕の数を増やしてみる

4.2 では，3 回分の試行を一度に行うことができた．5 回分，10 回分の試行を一度に行う関数 armselect5, armselect10 を作ってみよう．

## 5 製作対象 配列を使用したデータの受け渡し

4.2 では，3 つの情報を入力することができた．しかし，複数の腕の選択を考えると，

- 試行回数分だけ入力部分を書くのは面倒くさい
- 入力できる試行回数が固定となってしまう

という問題がある．そこで，配列を使ってデータの受け渡しをすることを考える．配列の場合の定義・実体は，5.1 プログラムの 10 行目および 42 行目からである．また，ポインタの演習 [3] で少し説明したように，配列の名前は特殊なポインタである．そのため，12 行目および 40 行目のような定義・実体の表し方も可能である．10 行目と 12 行目が同一であるということから，入力として渡している値は，ポインタ変数である．そのため，配列を使用した受け渡しである 32 行目をみると分かるように，関数の名前 (ポインタ) を入力としている．

注意点としては，配列を渡したからといって，何個の要素を持つ配列か (double array[10] なら 10 個の要素をもつ配列)，という情報は自動的に関数に渡らない．そこで，要素の数も入力として渡す必要がある．これが，10 行目 (12 行目) の定義中の num_try である．

## 5.1 プログラム

バンディット関係の関数は，ガイダンス資料 [1] に説明があるので一読すること．

```
1  /*********************************************************
2                    腕の一括選択 (配列)
3  *********************************************************/
4
5  #include <stdio.h>  /* システムの用意した入出力        */
6  #include <stdlib.h> /* システムの用意した便利関数      */
7  #include "bandit.h" /* ユーザが用意したバンデット関連 */
8
9  /* num_try 試行回分，腕を選ぶ (selection[i] で i+1 回目の腕*/
10 double armselect(int selection[], int num_try);
11 /* 違う定義の仕方
12 double armselect(int* selection, int num_try);
13 */
14
15 int main(){
16   int num_arm, loop, arm_select, selection[3];
17   double total_reward;
18
19   init_bandit();           /* バンディットの初期化 */
20
21   num_arm = get_arm_num();/* バンディットの腕の数を取得 */
22   printf("このバンディットの腕の数は%d です\n",num_arm);
23
24   /* バンディットの腕の選択 (3 試行分) */
25   for(loop=0 ; loop<3 ; loop++){
26     printf("バンデットの腕の番号を選択してください [1-%d]:",num_arm);
27     scanf("%d", &arm_select);
28     selection[loop] = arm_select;
29   }
```

```
30
31      /* 関数 armselect による一括選択 (3 回分) */
32      total_reward = armselect(selection, 3);
33
34      printf("総スコア:%lf\n",total_reward);
35
36      return 0;
37    }
38
39    /* 違う定義の仕方
40    double armselect(int* selection, int num_try){
41    */
42    double armselect(int selection[], int num_try){
43      int loop;
44      double total;
45
46      /* バンディット動作 num_try 回 */
47      total=0.0;
48      for(loop=0 ; loop<num_try ; loop++){
49        total = total + bandit(selection[loop]);
50      }
51
52      return total;
53    }
```

## 5.2 コンパイル

上記のプログラムを作成し，ファイル名 gameplay-arrayselect3.c として保存してコンパイルを行う．コンパイル後の名前を gameplay-arrayselect3 とする．

> gcc -o gameplay-arrayselect3 gameplay-arrayselect3.c bandit00.o

## 5.3 動作実験

作成したプログラムを動作させ，3 回腕が選択されていることを確認する．

## 5.4 やってみよう

### 5.4.1 ファイル分割

ファイル分割し，armselect 関数を arrayselect.h と arrayselect.c に記述し，実行ファイルを作成してみよう．

#### 5.4.2　選択できる腕の数を増やしてみる

5.1 では，3 回分の試行を一度に行うことができた．5 回分，10 回分の試行を一度に行うように変更してみよう．

#### 5.4.3　ポインタによるデータの受け渡し

5.1 の定義から，ポインタでの受け渡しも可能であることが分かる．ポインタを用いて同様にデータを受け渡すプログラムを作ってみよう．

# 6　製作対象 ポインタ・配列によるデータの出力

関数の定義をもう一度書くと、

- 出力の型　関数の名前 (入力の型および変数名, 入力の型および変数名, ...)
  - 例: int f(float a, double b);

である．述べたいポイントは，入力は複数個存在出来るのに，出力は 1 種類 1 個のみである不便さである．例えば先に作成した 3 回分試行を行う関数 armselect3 や armselect であるが，入力は選択する 3 回分の腕番号に対して，出力はそれぞれの入力に対して得られる報酬の合計のみである．もし，選択する 3 回分の腕番号を入力として，それぞれの入力に対して得られる報酬 3 つを出力としたい場合，どうしたらよいだろう．ここで，ポインタ・配列を使用すると上記を実現することが出来る．この他に構造体やファイルスコープ変数を用いた方法などあるが，ポインタ・配列を用いた方法が最も使用しやすく効率がよい．

さて，普通の変数を入力部分に置いた場合と，ポインタ・配列を入力部分に置いた場合のテストのプログラムが 6.1 である．入力部に通常の変数が入っているものが 26 行目である．これに対してポインタ・配列が入力部分にきているものが，35 行目と 42 行目と 52 行目である．まず，これを作成して実行してもらいたい．通常の変数では，関数の中で値を変更されても 25 行目と 27 行目で値は変わらない．それに対して，ポインタ・配列が入力にきている関数では，関数の中で値が変更されると，関数外でも値が変わってしまっていることが分かる (実行してみると分かる)．この性質を使うと，入力部分にあるポインタ・配列変数に，関数の中での処理結果を代入すると，その関数の出力として使えることが分かる．この場合であると，pointer 関数，pointerarray 関数では，入力部分の int* p に演算結果 (今回は単なる値の代入であるが) を代入することで，関数の出力として使うことが出来る．

## 6.1　プログラム

```
1   /*********************************
2                多出力関数
3   *********************************/
4
5   #include <stdio.h>
6   #include <stdlib.h>
```

```
7
8   int normal(int input);
9   int pointer(int* p);
10  int pointerarray(int* p);
11
12  int main(){
13    /* 通常変数 */
14    int input;
15    /* ポインタ変数 */
16    int *pointer1, *pointer2;
17    /* ポインタ変数 (配列) */
18    int *parray;
19    /* 配列 */
20    int array[10];
21
22
23    /* 通常変数の挙動 */
24    input = -1;
25    printf("input = %d\n",input);
26    normal(input);
27    printf("input = %d\n",input);
28
29    /* ポインタ変数の挙動 */
30
31    /* pointer1 は input と同一 */
32    pointer1 = &input;
33    *pointer1=-2;
34    printf("*pointer1 = %d\n",*pointer1);
35    pointer(pointer1);
36    printf("*pointer1 = %d\n",*pointer1);
37
38    /* pointer2 は単独で存在 */
39    pointer2 = malloc(sizeof(int));
40    *pointer2=-3;
41    printf("*pointer2 = %d\n",*pointer2);
42    pointer(pointer2);
43    printf("*pointer2 = %d\n",*pointer2);
44
45    /* ポインタ変数 (配列) の挙動 */
46    parray = malloc(sizeof(int)*10); /* 10 個の配列 */
47    /* parray[2] と parray[3] でテスト */
48    *(parray+2)=-4; /* parray[2] とも書ける */
49    parray[3]=-5;   /* *(parray+3) とも書ける */
```

```
50
51      printf("parray[2]:parray[3]=%d:%d\n",parray[2],parray[3]);
52      pointerarray(parray);
53      printf("parray[2]:parray[3]=%d:%d\n",parray[2],parray[3]);
54
55      /* 配列の挙動 */
56      /* array[2] と array[3] でテスト */
57      array[2]=-6;
58      array[3]=-7;
59
60      printf("array[2]:array[3]=%d:%d\n",array[2],array[3]);
61      pointerarray(array); /* 配列の名前は特殊なポインタ */
62      printf("array[2]:array[3]=%d:%d\n",array[2],array[3]);
63
64      free(pointer2);
65      free(parray);
66      return;
67    }
68
69    int normal(int input){
70      input = 100;
71
72      return 0;
73    }
74
75    int pointer(int* p){
76      *p = 200;
77
78      return 0;
79    }
80
81    int pointerarray(int* p){
82      p[2] = 300;
83      p[3] = 400;
84
85      return 0;
86    }
```

## 6.2 コンパイル

上記のプログラムを作成し，ファイル名 multioutput-samle.c として保存してコンパイルを行う．コンパイル後の名前を multioutput-sample とする．

> gcc -o multioutput-sample multioutput-sample.c

## 6.3 動作実験

作成したプログラムを動作させ，通常変更とポインタ・配列変数の挙動の違いを確認する．

## 6.4 調べてみよう

”なぜ”このようなことが起こるのか，教科書・ネットで調べたり友人と相談して考えてみよう．

## 6.5 やってみよう

下記は，3 回試行を一度に行うプログラムである．3 回分の試行を selection[3] に記憶したのち，play 関数によって一度にバンディットを試行している．9 行目の定義のように，入力変数として num_try:実行試行回数，selection:選択する腕番号，reward:各試行毎の獲得報酬を持ち，出力として総獲得報酬を持つ関数の実体を作成してみよう (39 行目から作ってみよう)．

```
1  /*********************************************************
2                  腕の一括選択 (他入出力関数)
3  *********************************************************/
4
5  #include <stdio.h>  /* システムの用意した入出力        */
6  #include <stdlib.h> /* システムの用意した便利関数      */
7  #include "bandit.h" /* ユーザが用意したバンデット関連 */
8
9  double play(int num_try, int* selection, double* reward);
10
11 int main(){
12   int num_arm, loop, arm_select, selection[3];
13   double total_reward, reward[3];
14
15   init_bandit();          /* バンディットの初期化 */
16
17   num_arm = get_arm_num();/* バンディットの腕の数を取得 */
18   printf("このバンディットの腕の数は%d です\n",num_arm);
19
20   /* バンディットの腕の選択 (3 試行分) */
21   for(loop=0 ; loop<3 ; loop++){
22     printf("バンデットの腕の番号を選択してください [1-%d]:",num_arm);
23     scanf("%d", &arm_select);
24     selection[loop] = arm_select;
25   }
26
27   total_reward = play(3, selection, reward);
28
```

```
29      /* 結果の表示 */
30      for(loop=0 ; loop<3 ; loop++){
31        printf("%d 回目:スコア=%lf\n",loop+1,reward[loop]);
32      }
33      printf("総スコア:%lf\n",total_reward);
34
35      return 0;
36    }
37
38    double play(int num_try, int* selection, double* reward){
39
40    }
```

# 7　参考文献

## 参考文献


[1] プログラミング演習～ガイダンス～

[2] プログラミング演習～配列～

[3] プログラミング演習～ポインタ～