基礎プログラミング演習 II 教材 (#8)

■ 関数

プログラムの一部分を機能ごとに切り出し、関数として分けて記述することができます。これによってより分かりやすいプログラムの記述が可能です。またプログラムを機能ごとの単位に仕分けて書くことによって、機能単位の再利用が容易になり、共通の関数を使った別のプログラムを簡単に作れるようになります。大規模なプログラムを作る重要な手法の一つです。
実際、C 言語では printf( ), scanf( ), sin( ) など標準的に多くの関数が用意されており、受講生は既にこれらを便利に利用しています。こうした関数を自分たちで定義することができるわけです。

□ 簡単な関数の例

以下に階乗 (factorial) を計算するプログラムの例を示します。サンプルを取得・実行して、たとえば 5 の階乗が正しく 120 (5 * 4 * 3 * 2 * 1) になることを確認して下さい。

入力処理

計算処理

出力処理

```
#include <stdio.h>
int main() {
  int n,i,fac;

  printf("n=");
  scanf("%d", &n);

  fac=n;
  for(i=n-1; i>1; i--) {
    fac*=i;
  }

  printf("factorial = %d\n", fac);

  return 0;
}
```

11 以上の階乗を求めようとすると桁あふれを起こして正しい結果が出ないので注意。

プログラムをよく見ると、入力処理、計算処理、出力処理に分けられることがわかります。
今回はこのうち計算処理の部分だけを別の関数にします。

階乗の計算処理だけを factorial という名前の関数に分けて書いた例を以下に示します。二つのプログラムの関係をメインルーチン、サブルーチンと表現することもあります。

（新しい数学関数（のようなもの）をひとつ作ることができたわけです！）

```
int main() {
  int n, fac;

  printf("n=");
  scanf("%d", &n);

  fac=factorial(n);

  printf("factorial = %d\n", fac);

  return 0;
}
```

呼び出し

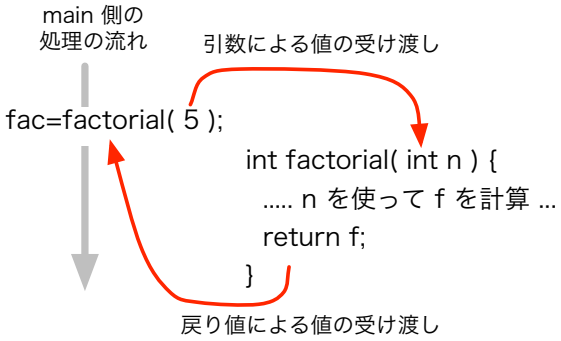
メインルーチン（main 関数）

```
int factorial(int n)
{
  int i,f;

  f=n;
  for(i=n-1; i>1; i--) {
    f*=i;
  }

  return f;
}
```

戻り

サブルーチン（factorial 関数）

特徴：
・関数には名前があり、その名前で呼び出す。
・関数の処理は関数名に続くブロック { } に記述される。



処理の流れ：
・ プログラムは main( ) の上から順に実行される。
・ main の途中で factorial( ) 関数を呼び出すと、
・ factorial 関数に処理が移る。
  ・ 引数によって値が渡され、
  ・ 階乗の計算を行い、
  ・ return でメインルーチンに処理が戻る
  ・ そのとき return f として結果を戻す
・ メインルーチン側で factorial( ) 関数の結果（戻り値）を fac 変数に代入。
・ そのまま main の処理を続行する。

> 階乗の定義から `i >= 1` とすべきかもしれませんが、結果が変わらないので、無駄な処理を省くために `i > 1` で済ませています。

□ 関数の定義、値の引き渡し方

例では factorial 関数は以下のようにして定義しました。

```
int factorial(int n)
{
  ……
}
```

> `int factorial(…)` の先頭の型 `(int)` は、`factorial` 関数の戻り値（後述）の型を示す。

この関数をメインルーチン側から呼び出すときは、例えば以下のようにします。

factorial(値);

値の部分には数値(100 等)や変数(n 等)、それらを組み合わせた計算式などが入ります。この値がサブルーチン側（factorial 関数側）で用意した変数 n に引き渡され、サブルーチン内での実行が始まります。こうした受け渡しに用いるカッコ内の値（や変数）を引数（ひきすう）と呼びます。

□ 戻り値

return 文の実行によって、処理はサブルーチンからメインルーチンに戻ります。このとき、「 return 値; 」と書くことで関数の結果そのものを設定することができます。
（数学関数がそのようなものだったことを思い出してください。）

つまり呼び出すときに fac=factorial(n); のようにしておくと、サブルーチン側の return によって戻された結果（ n の階乗）が変数 fac に代入されます。

注意：
return で戻す値の型は、関数自体の宣言文（ int factorial(… ）の部分)の先頭にある型宣言と一致していること。

□ 実際のプログラムの記述

実際の C プログラムのなかでは、サブルーチンはメインルーチンと同じファイルのなかに（縦に）並べて書きます。

サブルーチン（呼び出される側）はメインルーチンの呼び出しより前に記述されていなければなりません。

つまりサブルーチンは main より前（上）に書くことになります。

関数から関数を呼び出すこともできますが、そうした場合も常に呼ばれる側がより前（上）に記述されている必要があります。

```
#include <stdio.h>

int factorial(int n)
{
  int i,f;
  f=n;
  for(i=n-1; i>1; i--) {
    f*=i;
  }
  return f;
}

int main() {
  int n, fac;
  printf("n=");
  scanf("%d", &n);

  fac=factorial(n);

  printf("factorial = %d\n", fac);
  return 0;
}
```

□ main の意味

よく見ると main( ) もひとつの関数として機能している（それも整数型の）ことが分かると思います。実際、C 言語ではメインルーチンは特別な存在ではなく、単に実行時に最初に呼び出される関数がたまたま main( ) という名前に固定されている、というだけのことです。

□ 課題 1.

入力した二数の階乗の和を作るプログラムを作ってください。

サンプルプログラムの factorial 関数を二度使えば良いでしょう。機能単位に切り出した関数を再利用することでプログラムの機能修正が容易に、また読みやすくなることを感じてください。

□ 複数の引数・一般書式

関数は以下のようにして定義します。引数はカンマで区切って複数用意することができます。

型 関数名(型 変数 1, 型 変数 2, … 型 変数 z) {
　….
}

これをメインルーチン側は、**関数名(値 1, 値 2, ... 値 z)** として呼び出します。
引数は複数用意できますが、そのときは呼び出す側の用意した値と、呼ばれる側の変数が左から順に組み合わされて代入されます。両者で値と変数の数と型が一致していることが重要です。

上の例では、メインルーチンで引数に設定した「**値 2**」は、サブルーチンで用意した「**変数 2**」に設定されます。「**値 2**」のデータ型は「**変数 2**」の型に一致していなければなりません。

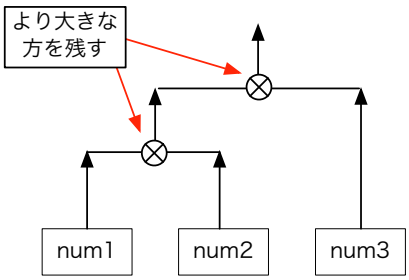
複数の引き数をもつ関数について教科書 p.231 「複数の引き数を持つ関数を使う」を参照してください。

□ 課題 2.

二つの数字を scanf で入力し、大きい方を出力（表示）するプログラムを作成してください。二つの引数を与えると、大きい方を戻り値に設定して返す関数を作って実現してくださいね。

□ 課題 3.

課題 2. で作成した「大きい方を戻り値に設定して返す」関数を利用して、三つの数字を入力して最も大きな数を出力するプログラムを作成してください。



□ 課題 4.

150 1.08 などと、単価と消費税率の二数を入力すると価格を出力するプログラムを作成してください。この課題も計算を行う関数を作って実現してください。(消費税率は関数内に固定的に書かず、引数として関数に渡すようにしてください。単価と税率の型の違いに注意。なお余り実用性の無いプログラムですが、練習課題なのでその点は気にしないで下さい。)

□ 課題 5.

入力した二数の階乗の和を作るプログラムを作ってください。ただし、課題 1. とは異なり、「二数を引数に与えると、それぞれの階乗を求め、その和を戻り値に設定して返す」関数を定義して実現するように修正してください。これは既に作成した階乗を求める関数を再利用する、という実験です。

「動けば良い」では無く、よりうまく関数を使って分かりやすい（見通しの良い）コードにする工夫を検討して下さい。

□ 引き数に対する値渡し

教科書 p.231 の図 8-7「値渡し」を参照。そこでは main( ) 関数内から buy( ) 関数を呼び出す際に実引数として値 20 が渡され、buy ( ) 関数のなかでは仮引数 x として受け取って処理されています。

先に示したサンプルプログラムでは fac=factorial(n); のように変数一つを与えて呼び出しました。
しかし呼び出し側で設定するのは「値」であり、常に変数一つだけを記述するわけではありません。例えば以下のすべての書き方があり得ます。

```
fac=factorial( 10 );  定数
fac=factorial( i );  変数
fac=factorial( i + 2 );  計算式
fac=factorial( factorial( 3 ) + 3 );  関数（自分自身かもしれない）を含む計算式
```

サンプルプログラムでは、呼び出し側と関数定義側で同じ名前の変数(n)を使っていますが、これは偶然で、違った名前で問題ありません。

```
int factorial(int n)
  .....
}

int main() {
  fac=factorial(n);
```

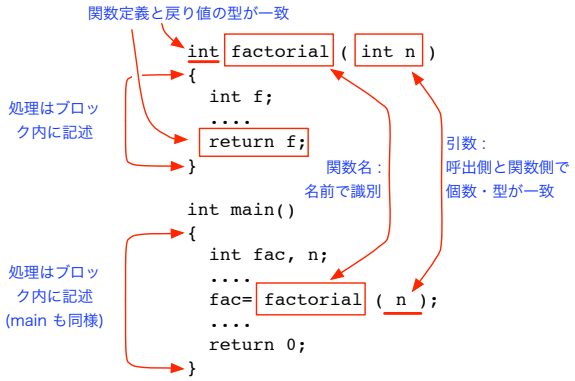
たまたま同一名だった

そもそもメイン側から引き渡されるのは変数ではなく「値」(変数だけでなく定数や式もあり得る）であることに注意してください。

（しつこいですが）良くある勘違い：
サブルーチン側で仮引数に用いた変数の値を変更しても、メインルーチン側で実引数に使った変数の値は変更されません。(実引数が a+2 の場合何が起きる？ a + b なら？と考えると解りやすい)

■ 関数の記述・構造（復習）

右にサンプルプログラムの構造を抜き出したものを示します。



・関数は main より前に書く
・main() は「main 関数」である
・関数の処理はブロック { } 内に書く
・関数には関数名が付く
・メイン側は関数名を用いて呼び出す
・引数の型と個数を両者で一致させる
・戻り値には型がある

「引数を受け取って処理が飛び込み、戻り値をもって帰る」関数の動作イメージがありますか？

■ 引数の無い関数

引数のある関数ばかりを使って説明してきましたが、引数は無くても構いません。

関数の定義では引数の列の代わりに void と書き、呼び出しの際は function( ) のように、関数であることを示すカッコだけを付けます。（右図参照）

```
int function( void ) {
  .....
  return 0;
}

int main( ) {
  .....
  a=function( );
  .....
```

■ 戻り値の無い関数

同様に戻り値がない関数もあり得ます。例えば引数の数値を出力するだけで戻り値が不要な場合です。

関数の定義では戻り値の型（関数自身の型）に void と書き、return には戻り値を置きません。（return は無くても、ブロックの最後に到達すれば関数からは戻ってくるため、return そのものを省略する場合も多い）

```
void function( int n ) {
  printf("n=%d\n", n);
  return;
}

int main( ) {
  .....
  function( 10 );
  .....
```

引数も戻り値もない関数は例えば以下のように定義され、呼び出されるでしょう。

定義：

```
void function( void ) {
    ....
    return;  <<-- 省略しても良い
}
```

呼出：

```
function( );
```

■ 機能部品として関数を作る

機能部品として関数を作り、それを使ってプログラムを書く訓練をします。
題材を最初にやった問題群から幾らか取ります。過去の自分のプログラムを取り出して、その一部分を関数化してください。(残っていない、やっていない人はゼロから書いて貰うしかありませんが、、、)

□ 課題 6. じゃんけん判定（#1 キックスタートの課題）

以下に判定処理を janken( ) 関数に取り出し、判定結果に基づいて反応するプログラム構造となるように修正した例を示します。今の時点では余りにも処理が簡単すぎて、これで「より良い」プログラムになったような気がしないでしょうが、ともあれこのようにするのだ、と思ってください。
これを真似て、自分の作ったじゃんけんプログラムの判定処理を、同様に関数化してください。
なお、今ならグーチョキパーの 1, 2, 3 は#define でマクロ定義すべきですから、是非そのように直して下さい。

```
int main() {
  int me, you;

  scanf("%d %d", &me, &you);

  if( me == you ) { // あいこだった
    printf("even\n");
  } else if( ( ( me == 1 ) && ( you == 2 ) ) // 自分がグー
        || ( ( me == 2 ) && ( you == 3 ) )    // チョキ
        || ( ( me == 3 ) && ( you == 1 ) ) ) { // パー
    printf("win\n");
  } else {
    printf("loose\n");
  }

  return 0;
}
```

```
int janken(int me, int you)
{
  if( me == you ) { // あいこだった
    return 1;
  } else if( ( ( me == 1 ) && ( you == 2 ) )  // 自分がグー
          || ( ( me == 2 ) && ( you == 3 ) )   // チョキ
          || ( ( me == 3 ) && ( you == 1 ) ) ) { // パー
    return 2;
  } else {
    return 3;
  }
}

int main() {
  int me, you;

  scanf("%d %d", &me, &you);

  switch ( janken( me, you ) ) {
    case 1:
      printf("even\n");
      break;
    case 2:
      printf("win\n");
      break;
    default:
      printf("loose\n");
      break;
  }

  return 0;
}
```

呼び出し

□ 課題 7. うるう年判定

うるう年判定も、そのための関数を使うように修正してください。
論理演算子を用いた判定ロジックだけを関数として取りだすように。出力処理などは main() 側で行うこと。

□ 課題 8. 素数判定

これも同様に素数判定関数を使うように修正してください。