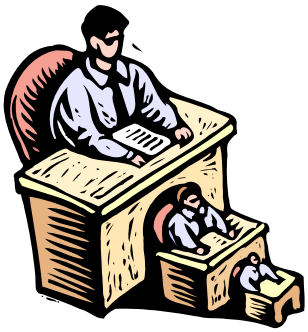
第１１回関数Ⅲ
（関数の応用）



1

今回の目標

- 副作用を持つ関数について理解する。
- 再帰的な関数定義について理解する。

☆階乗を求める関数を再帰的定義により記述する。

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

ライブラリ関数には、sqrt のように式の中で使う関数と、
printf のように単文として記述する関数がある。

4

通常の関数

数学関数などの普通の関数は、
仮引数に受け取った値を使って戻り値を計算することを目的とする。

関数の入力：実引数として与えられた値 （仮引数に受け取る）
関数の出力：戻り値 （関数呼び出しを含む式の中で利用する）

通常の関数の定義例：

```
/* 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;
}
```

戻り値の計算に必要な処理だけを行う

通常の関数の利用例：

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

式の中に関数呼び出しを書く

5

副作用を持つ関数

仮引数に受け取った値を使って戻り値を計算する処理以外の処理を
関数の副作用と呼ぶ。
仮引数や戻り値を持たない関数でも、副作用によって外部とやりとりを行える。
（普通の関数は副作用を持たない。）

副作用の例： 標準入出力を利用する副作用

```
int  scanInt()
{
   int   input;
   scanf("%d", &input);
   return   input;
}
```

標準入力から値を読み込む副作用

```
void   printInt(int  x)
{
   printf("%d¥n", x );
   return ;
}
```

標準出力へ値を出力する副作用

6

戻り値の無い関数

戻り値の計算を目的としないような関数を書くこともできる。

戻り値の無い関数の定義例：

void という特別な型を指定

```
void   printInt(int  x)
{
   printf("%d¥n", x );
   return ;
}
```

return の後には式を書かない

戻り値の無い関数の利用例：

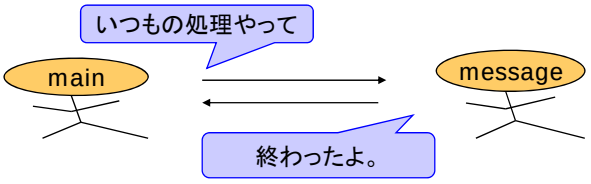
```
printInt(100);
```

関数呼び出しを単文として書く

7

プログラム例1： 副作用を利用した関数の練習

```
/* side_effect.c  副作用を利用した関数の練習  コメント省略 */
#include<stdio.h>
void  message ();
int main()
{
   message( );
   message( );
   return  0;
}

void  message()
{
   printf("こんにちは！¥n");
   return ;
}
```



8

main関数

mainも副作用を持つ関数の一つ。

```
int  main()
{
    int  age;      /*年齢*/

    printf("年齢は？¥n");
    scanf("%d",&age);
    printf("年齢は %d  歳です。¥n",age);

    return 0;
}
```

main関数の中でだけ利用できる変数の宣言

このプログラムで行う処理の内容（標準入出力などの副作用を含む）

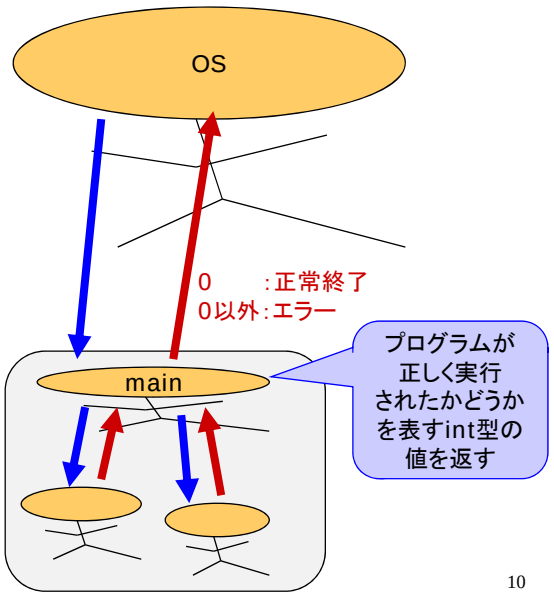
mainは特別な関数で、一つのプログラムに必ず1つだけなければいけない。
プログラムの実行はmain関数の最初から行われる。

9

関数mainの型とOS

```
/* ～.c  */

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

main関数は、
OSとのやりとりを
司る大元の関数。
プログラムに必ず1つ
しかも1つだけ存在する。



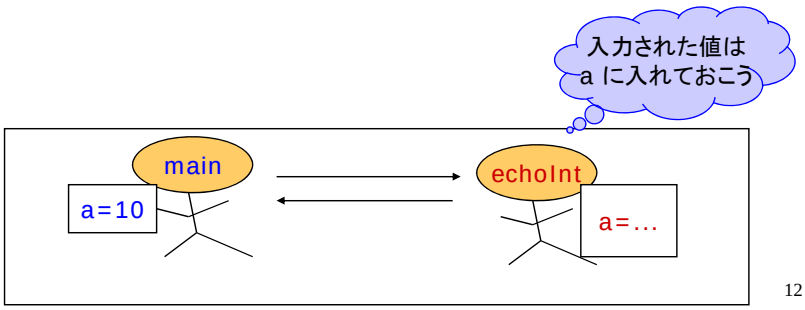
10

プログラム例2： 変数有効範囲の確認（復習）

```
/* test_scope2.c  変数有効範囲の確認  コメント省略 */
#include<stdio.h>
void  echoInt();

int  main()
{
   int  a;

   a = 10;
   printf(“echoInt()実行前¥n”);
   printf(“mainの変数aの値は %d です。¥n",a);
   echoInt();
   printf(“echoInt()実行後¥n”);
   printf(“mainの変数aの値は %d です。¥n",a);

   return  0;
}

/*  次に続く  */
```

11

```
/*  続き   */
void  echoInt()
{
   int a;

   printf(“整数値を入力してください。 ");
   scanf("%d", &a);
   printf(“入力された値（変数aの値）は %d です。¥n", a);

   return ;
}
```



12

再帰

関数が自分自身を呼び出す事。
C言語では、再帰的な関数を扱える。

書式:

```
戻り値の型　　関数名（仮引数宣言, … ）
{
  if ( 条件 )
    {
      return　この関数を利用せずに戻り値を計算する式;
    }
  else
    {
      return　この関数を再帰的に利用して、戻り値を計算する式;
    }
}
```

特別な値が実引数に与えられた場合は、自分自身を呼び出さない。（再帰の基礎）

それ以外の値が実引数に与えられた場合には、自分自身を呼び出す式を使って戻り値を計算する。

13

プログラム例3の原理：階乗 $n!$の2つの数学的表現

（1）繰り返しによる表現

$$n! = 1 \times 2 \times \cdots \times i \times \cdots \times n$$

$$= \prod_{i=1}^{n} i$$

（上記は $n \geqq 1$ のとき。0! は1 。）

（2）漸化式による表現

$$n! = \begin{cases} 1 & n = 0 \text{ のとき} \\ n \times (n-1)! & n \geq 1 \text{ のとき} \end{cases}$$

14

再帰関数の定義例

```
int    factorial(int n)
{
   if (n==0)
     {
      return   1 ;
     }
   else
     {
      return  n * factorial(n-1);
     }
}
```

0!=1 なので、
実引数が 0 の場合は
自分自身を呼び出さない。
（再帰の基礎）

0以外の値が実引数に
与えられた場合には、
$n! = n \times (n-1)!$
であることを利用して階乗を求める。
（関数の再帰呼び出し）

漸化式をそのままプログラムにできる。

$$n! = \begin{cases} 1 & n = 0 \text{ のとき} \\ n \times (n-1)! & n \geq 1 \text{ のとき} \end{cases}$$

15

再帰の典型的な書き方

書式だけを抽出

```
int   関数名(int n)
{
   if(n==0)
     {
      /*再帰関数の基礎*/
      return  ????;
     }
   else
     {
      /*再帰部分*/
      return   関数名(n-1);
     }
}
```

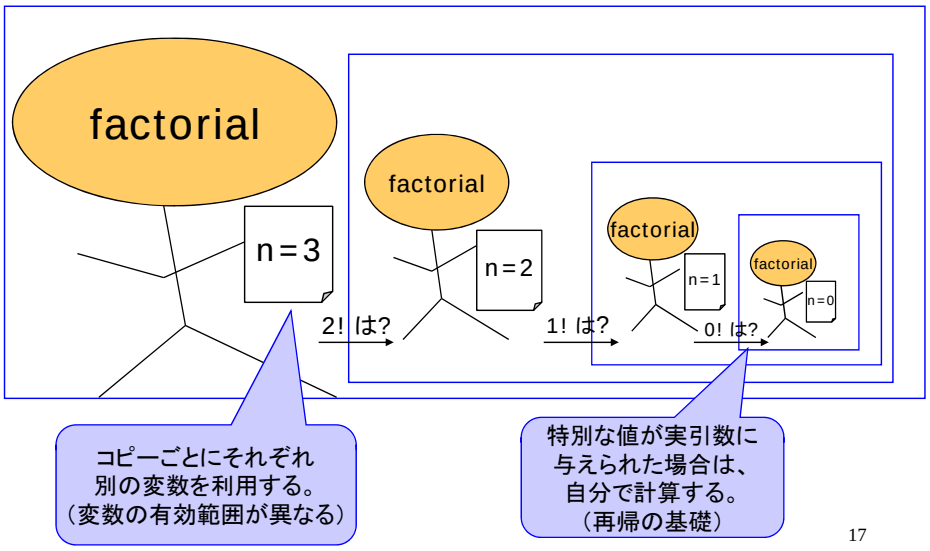
必ず「再帰の基礎」（出口）
を作ること。

再帰呼び出しは、
必ず出口に近づくようにすること。
（n が正の整数ならば、
n よりは n-1 のほうが0に近い）

16

イメージ

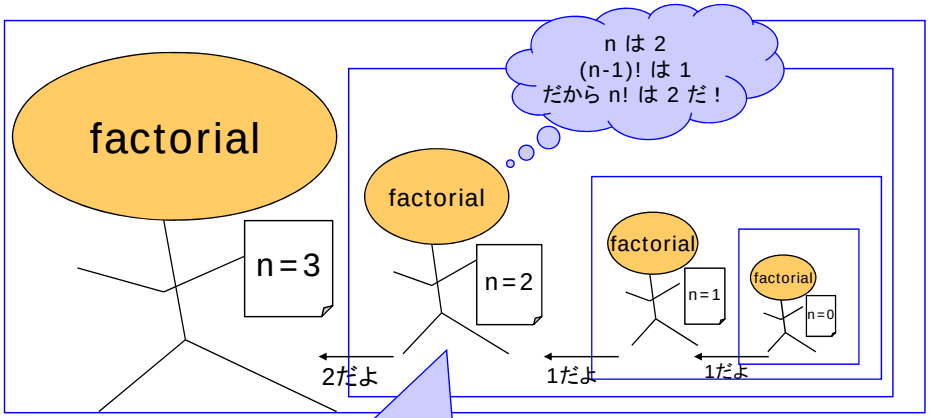
再帰関数は、自分自身のコピーに仕事を押し付ける。



17

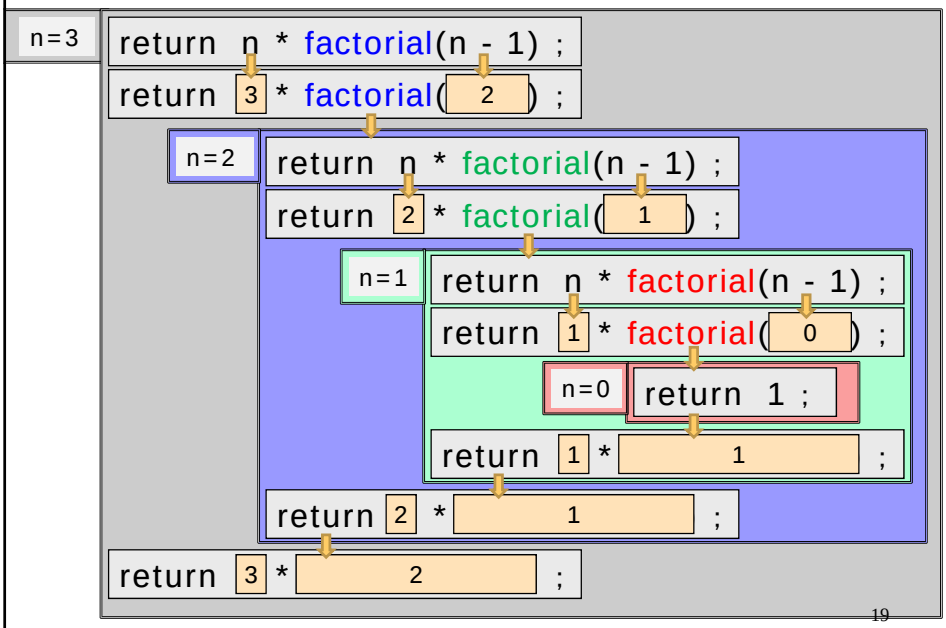
イメージ

再帰関数は、自分自身のコピーに仕事を押し付ける。



18

再帰的関数呼び出しにおける計算



19

終わらない再帰関数

再帰関数は、必ず基礎（出口）部分に辿りつけないといけない。

再帰の基礎部分が無い再帰関数

```
/*終わらない再帰関数1*/
int    factorial(int n)
{
   return   n * factorial(n-1);
}
```

NG

再帰関数は、ちょっと注意を怠ると、すぐに終わらないプログラムになってしまうので注意する事。

プログラムが終わらないときには、プログラムを実行しているktermで、コントロールキーを押しながら、cキーを押す。（C－c）

出口に近づかない再帰関数

```
/*終わらない再帰関数2*/
int    factorial(int n)
{
   if (n==0)
     {
      return  1;
     }
   else
     {
      return  n * factorial(n);
     }
}
```

NG

20

プログラム例3：階乗を求める再帰的関数の定義と利用

```
/*
   作成日：yyyy/mm/dd
   作成者：本荘太郎
   学籍番号：B0OB0xx
   ソースファイル：fact_rec.c
   実行ファイル：fact_rec
   説明：階乗を求める再帰的関数を用いて、与えられた値の階乗を計算する。
   入力：標準入力から0以上15以下の整数nを入力。
   出力：標準出力にn!の値を出力。n!は正の整数である。
*/
#include<stdio.h>

/*プロトタイプ宣言*/
int  factorial(int n);       /*階乗n!を求める再帰的な関数*/
int  scanSmallInt(int  limit);  /* 標準入力から小さな非負整数を読み込む関数 */

int    main()
{
   int    n;       /* 階乗を求めるべき値（入力）*/
   int    fac;     /* nの階乗の計算結果（n!）*/
```

21

```
/*  次のページに続く  */
```

```
/* 続き */

   printf(“入力された値nの階乗 n! を計算します。¥n");
   printf("n=? ¥n");
   n = scanSmallInt(15);

   /* 入力値チェック */
   if(n==-1)
     {
       printf(“nには0から15までの整数を入力してください。¥n");
       return  -1;
     }
   /* これ以降ではnは0から15までの整数 */
   /* nの階乗の計算 */
   fac = factorial(n);

   printf("%d!  =  %10d ¥n", n, fac);

   return 0;
}
/* main関数終了 */

/*  次のページに続く  */
```

22

```
/* 続き */
/*
   階乗を求める再帰的関数
   仮引数 n : 階乗を求める値（0以上15以下の整数値とする。）
   戻り値    : nの階乗(正の整数値)を返す。
*/
int   factorial(int n)
{
   /* 漸化式に基づく、階乗の再帰的定義 */
   if (n==0)
     {
      /*再帰の基礎。0! = 1 であることを利用。*/
      return  1 ;
     }
   else
     {
      /* 再帰呼び出し。n! = n*(n-1)!  であることを利用。 */
      return  n * factorial(n-1);
     }
}
/* 関数factorialの定義終了 */

/*  次のページに続く  */
```

23

```
/* 続き */
/*
   標準入力から小さな非負整数を読み込む関数
   仮引数 limit : 入力値として許す整数の上限値(正の整数)
   戻り値        : 入力された値が0以上limit以下ならばその値
                   (0以上limit以下の整数)。そうでなければ、-1を返す。
*/
int   scanSmallInt(int  limit)
{
   int   input;  /* 標準入力から入力された(0以上limit以下の)値 */

   scanf("%d", &input);
   if ( 0<=input && input <= limit )
     {
      return  input;
     }
   else
     {
      return  -1;
     }
}
/* 関数scanSmallIntの定義終了 */

/* 全てのプログラム(combi.c)の終了 */
```

24

プログラム例3の実行結果

```
$ ./fact_rec
入力された値nの階乗 n! を計算します。
n=?
5
5!  =        120
$
```

25