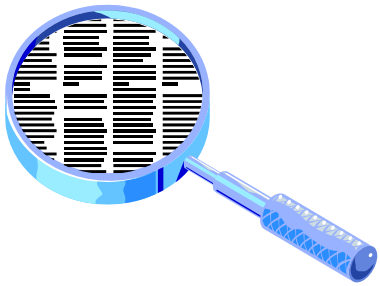
第10回関数 Ⅱ
（ローカル変数とスコープ）



1

今回の目標

- 戻り値の計算に条件分岐や繰り返し処理を必要とするような、複雑な定義を持つ関数について理解する。
- 関数のローカル変数とスコープの役割について理解する。

☆階乗を求める関数を利用して、組み合わせの数を求めるプログラムを作成する

2

組み合わせの数を求める公式

$$ {}_nC_m = \frac{n!}{m! \times (n-m)!} $$

階乗の計算が何度も出てくる。
階乗を計算する関数があれば、
組み合わせの数は簡単に計算できる。

3

階乗を求める関数の定義（失敗例）

$$ x! = 1 \times 2 \times \cdots \times x $$

だけど・・・

NG

```
/* 実引数の階乗を求める関数の定義 */
int   factorial(int  x)
{
   return  1 * 2 * 3 * ・・・ * (x-1) * x  ;
}
```

C言語では、「・・・」を使って
式を省略することはできない。

場合分けを含む関数の定義

実引数にどのような値が与えられたかによって
戻り値を計算する式が異なるような関数を定義できる。

書式:

```
戻り値の型     関数名(仮引数宣言, ... )
{
   if ( 条件 )
     {
       return   条件が真のときの戻り値を計算する式;
     }
   else
     {
       return   条件が偽のときの戻り値を計算する式;
     }
}
```

仮引数を利用した条件式

場合分けを含む関数定義の例

$$\max 2(x, y) = \begin{cases} x & x > y \text{ のとき} \\ y & \text{それ以外} \end{cases}$$

```
/* 二つの実引数のうち、大きいほうの値を求める関数の定義 */
double   max2(double  x, double y)
{
   if ( x>y )
     {
       return   x;
     }
   else
     {
       return   y;
     }
}
```

x>y が真の場合に、
max2の戻り値を求める式

x>y が偽の場合に、
max2の戻り値を求める式

練習1

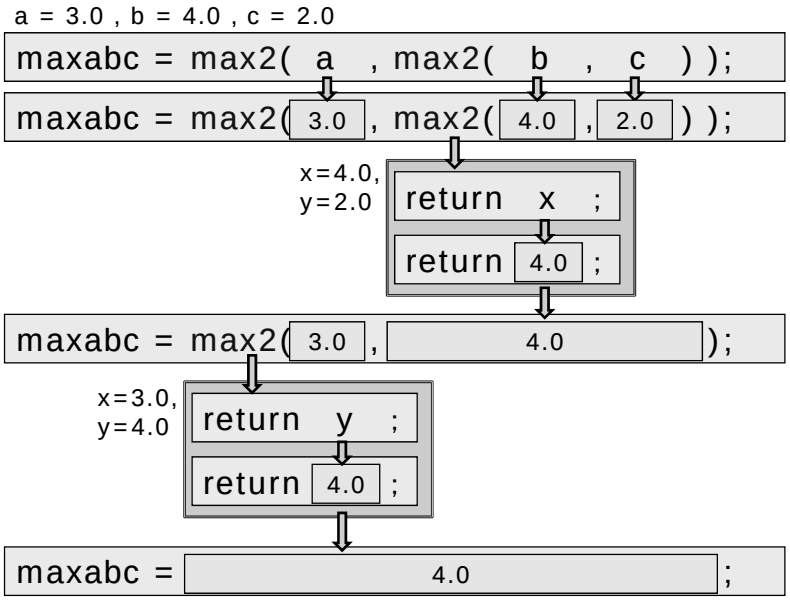
```
/*  条件分岐関数実験   max2.c   コメント省略        */
#include <stdio.h>

/* 関数のプロトタイプ宣言 */
double  max2(double  x, double  y) ;
/*二つの実引数のうち、大きいほうの値を求める関数*/

int  main()
{
   double  a;     /* 1つめの入力値  */
   double  b;     /* 2つめの入力値  */
   double  c;     /* 3つめの入力値  */
   double  maxabc;     /* a,b,cのうち最大の値  */

   printf("a?¥n");
   scanf("%lf",  &a);
   printf("b?¥n");
   scanf("%lf",  &b);
   printf("c?¥n");
   scanf("%lf",  &c);

/* 続く */
```

7

```
/*  続き */

   maxabc = max2(a, max2(b,c) );

   printf("a,b,c の最大値:%f¥n",  maxabc);

   return  0;
}


/* 二つの実引数のうち、大きいほうの値を求める関数の定義 */
double   max2(double  x, double y)
{
   if ( x>y )
     {
      return   x;
     }
   else
     {
      return   y;
     }
}
```

8

関数呼び出しを含む式の計算



ローカル変数を持つ関数の定義

ローカル変数を利用することで、
戻り値の計算を手順に分解して記述できる。

書式:

```
戻り値の型　　関数名（仮引数宣言, ... ）
{
   ローカル変数の宣言

   return　戻り値を計算する式;
}
```

ローカル変数:
この関数の戻り値を
計算するのに必要な
計算途中の値などを
入れておくための変数

ローカル変数を利用した
計算手順

仮引数の値と
計算されたローカル変数の値を使って
戻り値を計算する「戻り値の型」の式

ローカル変数を持つ関数定義の例

```
/* 点 (x1,y1) と点 (x2,y2) の間の距離を求める関数の定義 */
double  distance(double x1, double y1, double x2, double y2)
{
   double  x;       /* 二点のx座標の差 */
   double  y;       /* 二点のy座標の差 */
   double  sqsum; /* 座標の差の二乗和 */

   x = x2 – x1;
   y = y2 – y1;
   sqsum = square(x) + square(y);

   return  sqrt(sqsum) ;
}
```

ローカル変数の宣言

ローカル変数を利用した計算手順

11

練習2

```
/*  関数実験   lineseg2.c   コメント省略          */
/*     数学関数を用いるので、-lmのコンパイルオプションが必要 */
#include <stdio.h>
#include <math.h>

/* 関数のプロトタイプ宣言 */
double  square(double  x) ; /* 実引数を2乗した値を求める関数 */
double  distance(double x1, double y1, double x2, double y2) ;
/* 点 (x1,y1) と点 (x2,y2) の間の距離を求める関数 */

int  main()
{
   double  p_x;    /* 点pのx座標  */
   double  p_y;    /* 点pのy座標  */
   double  q_x;    /* 点qのx座標  */
   double  q_y;    /* 点qのy座標  */
   double  length; /* 線分pqの長さ */

   printf("点pの座標?¥n");
   scanf("%lf",  &p_x);
   scanf("%lf",  &p_y);
   printf("点qの座標?¥n");
   scanf("%lf",  &q_x);
   scanf("%lf",  &q_y);

/* 続く */
```

12

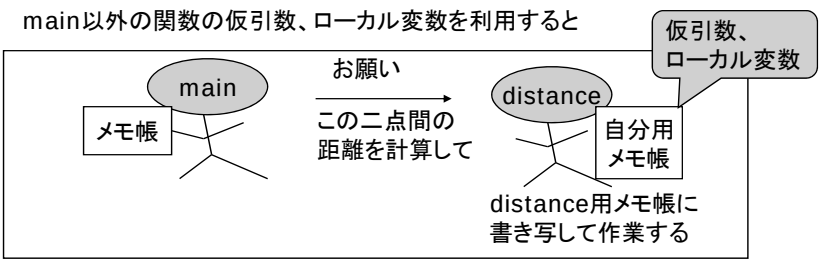
```
/* 続き */

   length = distance(p_x, p_y, q_x, q_y);
   printf("線分pqの長さ:%f¥n", length);
   return 0;
}

/* 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 ;
}

/* 点 (x1,y1) と点 (x2,y2) の間の距離を求める関数の定義 */
double  distance(double x1, double y1, double x2, double y2)
{
   double x;       /* 二点のx座標の差 */
   double y;       /* 二点のy座標の差 */
   double sqsum; /* 座標の差の二乗和 */

   x = x2 – x1;
   y = y2 – y1;
   sqsum = square(x) + square(y);

   return  sqrt(sqsum) ;
}
```

13

イメージ

以前は、main関数で宣言した変数しかなかった。

main以外の関数の仮引数、ローカル変数を利用すると



関数はローカル変数と仮引数のみを使って作業を行う。
間違ってmainの変数を書き換えてしまうのを防げる。

14

ローカル変数のスコープ（有効範囲）

関数定義の書式：

```
戻り値の型　　関数名（仮引数宣言，... ）
{
    ローカル変数の宣言

    return　戻り値を計算する式；
}
```

関数定義の仮引数や、関数定義内で宣言したローカル変数は、
宣言した関数定義の内部だけで有効。

したがって、異なる2つの関数の定義で同じローカル変数名を用いても、
それぞれの関数内で別々の変数としてあつかわれる。
（main関数で宣言した変数とも区別される）

変数のスコープ（有効範囲）

```
int  main()
{
    main関数内の変数宣言
    *****
    return  0;
}
```

main関数で
宣言した変数の
有効範囲

```
戻り値の型　　関数1（仮引数宣言，... ）
{
    ローカル変数の宣言
    *****
    return  ～ ；
}
```

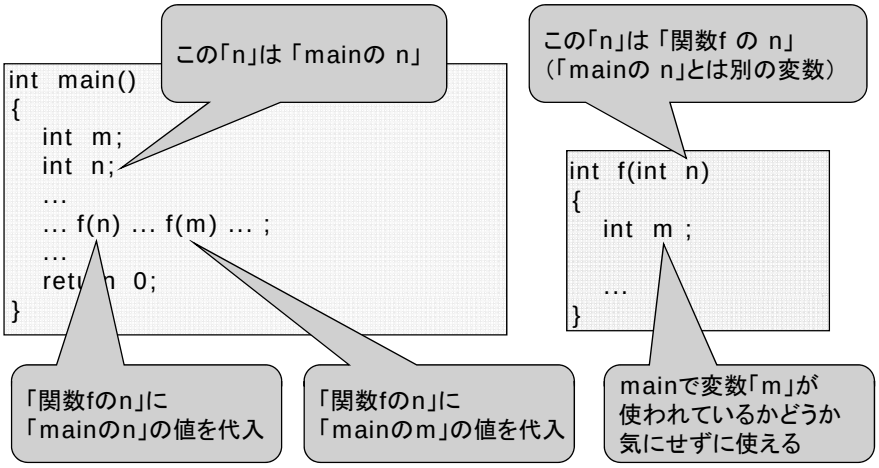
関数1の
仮引数、ローカル変数の
有効範囲

```
戻り値の型　　関数2（仮引数宣言，... ）
{
    ローカル変数の宣言
    *****
    return  ～ ；
}
```

関数2の
仮引数、ローカル変数の
有効範囲

スコープ（有効範囲）の利用

変数のスコープを利用すると、
他の関数の定義で使われている変数や仮引数と区別できる。



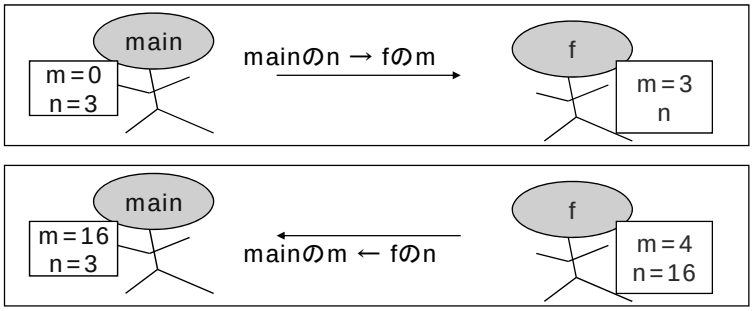
17

練習３

```
/* スコープ実験 test_scope.c コメント省略*/
#include <stdio.h>
int f(int m);

int main()
{
   int m;
   int n;

   m = 0 ;
   n = 3 ;
   printf("（mainの）m = %d ¥n", m);
   printf("（mainの）n = %d ¥n", n);

   m = f( n );

   printf("（mainの）m = %d ¥n", m);
   printf("（mainの）n = %d ¥n", n);

   return 0;
}

/* 次に続く */
```

18

```
/* 続き */
int f(int m)
{
   int n ;

   m = m + 1;
   n = m * m;

   return n;
}
```



19

繰り返しを含む関数の定義

ローカル変数を利用することで、
戻り値の計算式を、繰り返し処理を含む手順に分解して記述できる。

書式:

```
戻り値の型     関数名(仮引数宣言, ... )
{
   ローカル変数の宣言

   while (...)
    {
      ...
    }

   return   戻り値を計算する式;
}
```

ローカル変数:
この関数の戻り値を計算するのに必要な計算途中の値などを入れておくための変数

ローカル変数を利用した繰り返し処理(while や for)を含む計算手順

仮引数の値と
計算されたローカル変数の値を使って
戻り値を計算する「戻り値の型」の式

20

繰り返しを含む関数定義の例

```
/* 実引数の階乗を求める関数の定義 */
int  factorial(int n)
{
   int   i;      /* ループカウンタ */
   int   fact;  /* 1からiまでの積（iの階乗）*/

   fact = 1;
   for ( i = 1; i <= n ; i++ )
     {
       fact = fact * i ;
     }

   return   fact ;
}
```

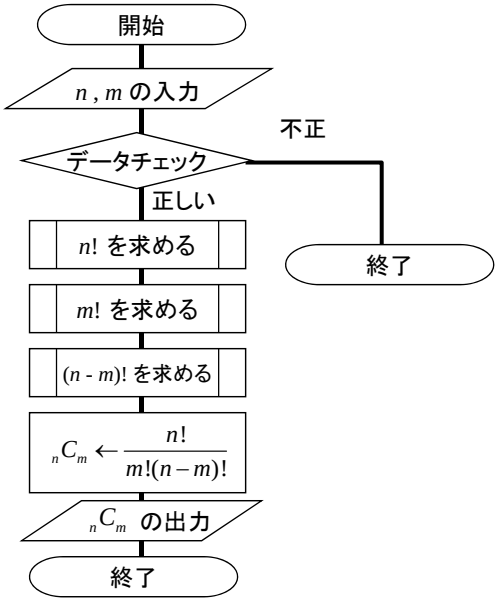
ローカル変数の宣言

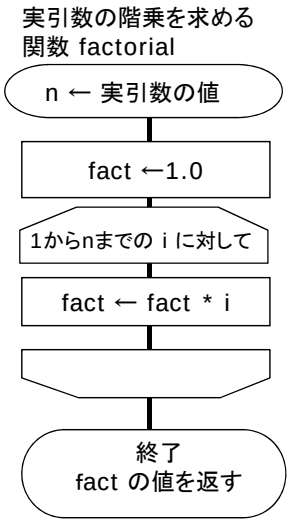
ローカル変数を利用した繰り返し計算手順

21

関数を利用したプログラムのフローチャート

組み合わせの数 ${}_nC_m$ を求める





22

組み合わせの数を求めるプログラム

```
/*
   作成日:yyyy/mm/dd
   作成者:本荘 太郎
   学籍番号:B00B0xx
   ソースファイル:combi.c
   実行ファイル:combi
   説明:組み合わせの数 nCm を求めるプログラム。
   入力:標準入力から2つの正の整数 n,m を入力。(n,mともに15以下とする)
   出力:標準出力に組み合わせの数 nCm を出力。組み合わせの数は正の整数。
*/
#include <stdio.h>

/* プロトタイプ宣言*/
int   factorial(int  n);   /* 階乗を計算する関数 */

int   main()
{
   /*変数宣言*/
   int   n;        /*nCm の n*/
   int   m;        /*nCm の m*/
   int   com;      /*組み合わせの数 nCm*/

/*  次に続く */
```



```
/* 続き */

   printf("組み合わせの数 nCm を計算します。¥n);
   printf("Input  n=? ");
   scanf("%d", &n);
   printf("Input  m=? ");
   scanf("%d", &m);

   /* 入力値チェック */
   if ( n<0 || 15<n || m<0 || 15<m || n<m )
     {
      /*不正な入力のときには、エラー表示してプログラム終了*/
      printf("不正な入力です。¥n");
      return  -1;
     }
   /* 正しい入力のとき、これ以降が実行される。*/

   /* 組み合わせの数を計算 */
   com = factorial(n) / ( factorial(m)*factorial(n-m) );

   printf("%d  C %d = %5d¥n", n, m, com);

   return 0;
}
/* main関数終了 */

/*  次に続く */
```

```
/* 続き */

/*
   階乗を求める関数
   仮引数 n : 階乗を求める値（0以上15未満の整数値とする。）
   戻り値   : nの階乗(正の整数値)を返す。
*/
int   factorial(int n)
{
   /*  ローカル変数宣言  */
   int   i;            /*ループカウンタ*/
   int   fact;         /* 1からiまでの積(iの階乗）*/

   fact = 1;           /* 0の階乗=1  であるので1を代入*/
   for(i=1; i<=n; i++ )
     {
        fact = fact * i ; /* 1からiまでの積 = (1から(i-1)までの積)×i */
     }

   /* 関数 factorial のローカル変数 fact の値(1からnまでの積)を戻す */
   return  fact;
}
/* 関数factorialの定義終 */
/* 全てのプログラム(combi.c)の終了 */
```


実行例

```
$make
gcc  combi.c  -o combi
$ ./combi
組み合わせの数 nCm を計算します。
Input  n=? 4
Input  m=? 3
4C3  = 4
$
```

```
$./combi
組み合わせの数 nCm を計算します。
Input  n=? 4
Input m=? 5
不正な入力です。
$
```