第9回関数と記憶クラス

1

今回の目標

- C言語における関数を理解する。
- 関数における仮引数の役割について理解する。
- 関数の戻り値について理解する。
- 関数の副作用について理解する。
- 変数の適用範囲（スコープ）について理解する。

☆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

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

3

関数の定義

書式:

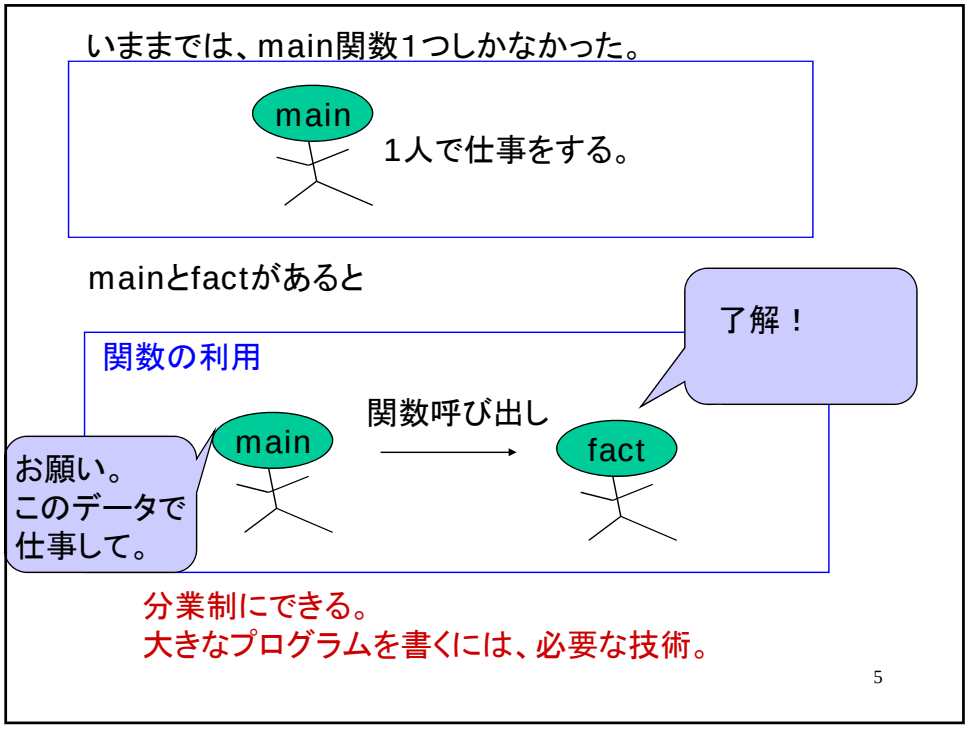
```
戻り値の型　　関数名（仮引数の宣言付きリスト）
{
        関数の本体
        return　式;
}
```

return文の式と同じ型

戻り値という。

もちろん、
定数や変数であっても
良い。

4



処理の分割と関数の利用

```
int main()
{
/*組み合わせ数を計算 */
   bunshi=fact(n);
   bunbo1=fact(m);
   bunbo2=fact(n-m);
   com=bunshi/(bunbo1*bunbo2);
   return 0;
}
```

階乗を求める専門家（関数）があると、便利。

```
int fact(int k)
{
   階乗の処理

   return  kの階乗;
}
```

依頼データを元に、結果を返す。（階乗の計算を行なう。）

関数例

mainというのも関数の一つ。
Cではプログラムは関数の集まりで作られる。

```
int main()
{
        関数の本体
        return 0;
 }
```

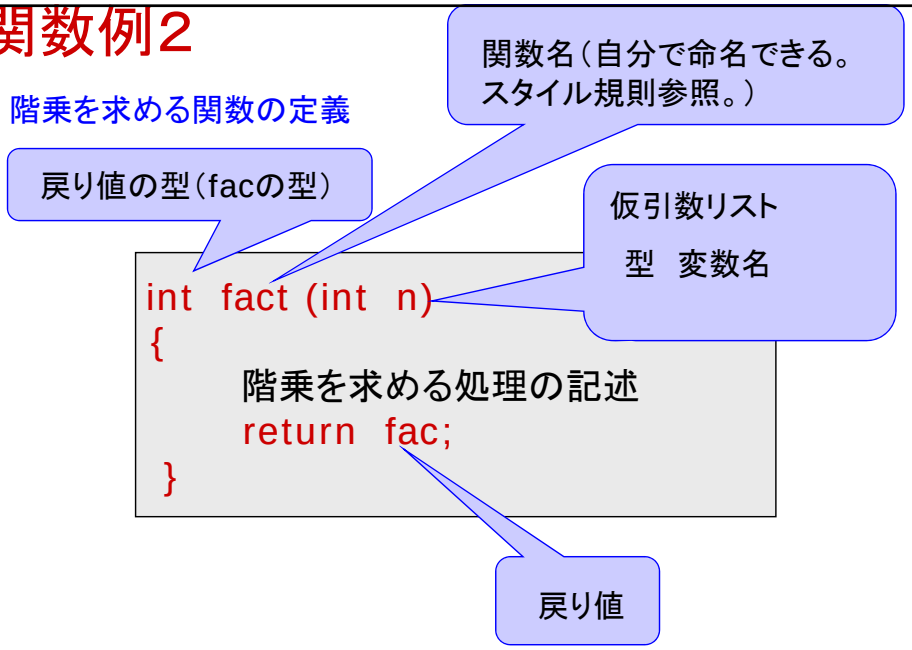
戻り値の型や関数への仮引数のリストは省略可能だが、
括弧の省略はできない。

mainは特別な関数名で、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。

7

関数例2

階乗を求める関数の定義



注意：メイン関数以外はプロトタイプ宣言を行うこと。

8

戻り値の型とreturn文

書式：

```
return　式;
または、
return ;
```

関数の実行中にreturn文に出会うと、その式の値を呼び出した関数に返してその関数を終了する。

```
int   fact(int n)
{
      ****
      int   fac;
      *****
      ******
      return  fac;
}
```

9

プロトタイプ宣言と関数の定義

書式

```
/*    プロトタイプ宣言      */
型1 関数名1(型a 仮引数a);          /*関数1のプロトタイプ宣言*/

/* メイン関数*/
int  main()
{
        *****
        return  0;
}

/*    関数1の定義(関数1の本体) */
型1  関数名1(型a   仮引数a)
{
        ****
        return    型1の式;
}
```

プロトタイプ宣言と関数定義において、セミコロンの有無に注意すること。

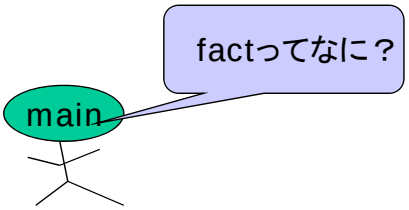
注意：メイン関数以外は、プロトタイプ宣言をメイン関数前に記述する。

10

プロトタイプ宣言の役割

プログラムは、
上から下に実行されるので、
プロトタイプ宣言が無いと。

```
/**/
int main()
{

        fact(m);
        return   0;
}

int fact(int n)
{
        return  fac;
}
```

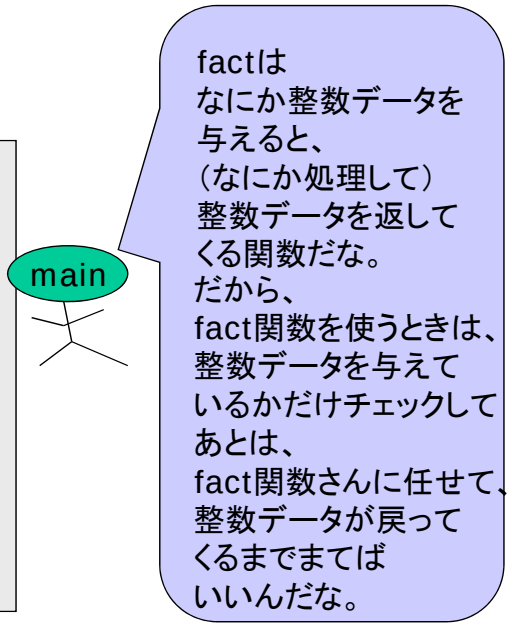


11

プロトタイプ宣言の役割2

プロトタイプ宣言があると。

```
int fact (int  n);
int main()
{

        fact(m);
        return 0;
}

int fact(int n)
{
        return   fac;
}
```



プロトタイプ宣言例

書式だけ抽出

```
int  fact(int  n);
int main()
{
        fact(m);
        return   0;
}

int fact(int  n)
{
        int fac;


        return  fac;
}
```

プロトタイプ宣言

関数の呼び出し

関数定義
（関数の本体）

13

実引数と仮引数

```
int  fact(int  n);
int main()
{
        fact(m);
        return   0;
}

int fact(int  n)
{
        int fac;


        return  fac;
}
```

呼び出す側の式（値）を
実引数（じつひきすう）と呼び、
呼び出される側の変数を
仮引数（かりひきすう）と呼ぶ。

このmの値は実引数

この変数nは仮引数

注意：実引数が変数の場合でも、
実引数と仮引数の名前
は異なっていてもかまわない。

14

関数へ値の渡し方

呼び出す方では、

関数名(式);　　や　　変数＝関数名(式);

などで関数を呼び出す。

呼び出される方では、
仮引数に実引数の値が”代入”される。

注意:実引数は変数でも定数でも式でもよい。

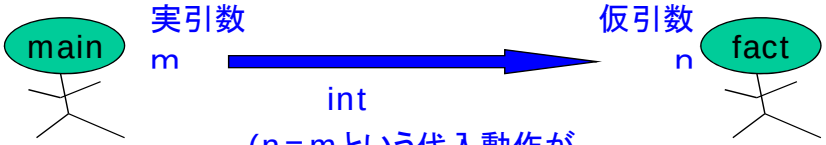
```
int  main()
{
        fact(m);
}
int     fact(int n)
{
        int fac;
        return    fac;
}
```

mainのmの値が、factのnに代入される。

15

```
int  main()
{
        fact(m);

        return    0;
}
```

```
int     fact(int n)
{
        return  fac;
}
```



(n＝mという代入動作が行われる。)

16

呼び出し側への戻り値の渡し方

呼び出す方で、単に 関数名(式); とすると、
せっかくの戻り値が利用できない。

変数＝関数名(式); とすると、戻り値が変数に代入される。

```
int  main()
{
        int a;
        a=fact(m);
}

int     fact(int n)
{
        int     fac;
        return fac;
}
```

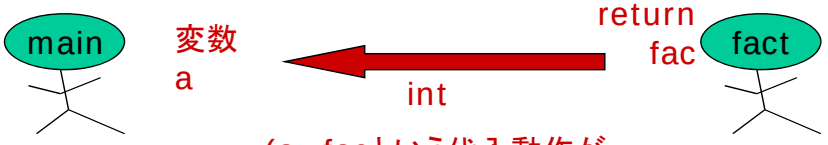
呼び出す側では、
"関数名(式)"全体を
一つの式あるいは一つの変数
のように考えてもよい。

factのfacの値が、
mainのaに代入される。

17

```
int  main()
{
        int a;
        a=fact(m);

        return  0;
}
```

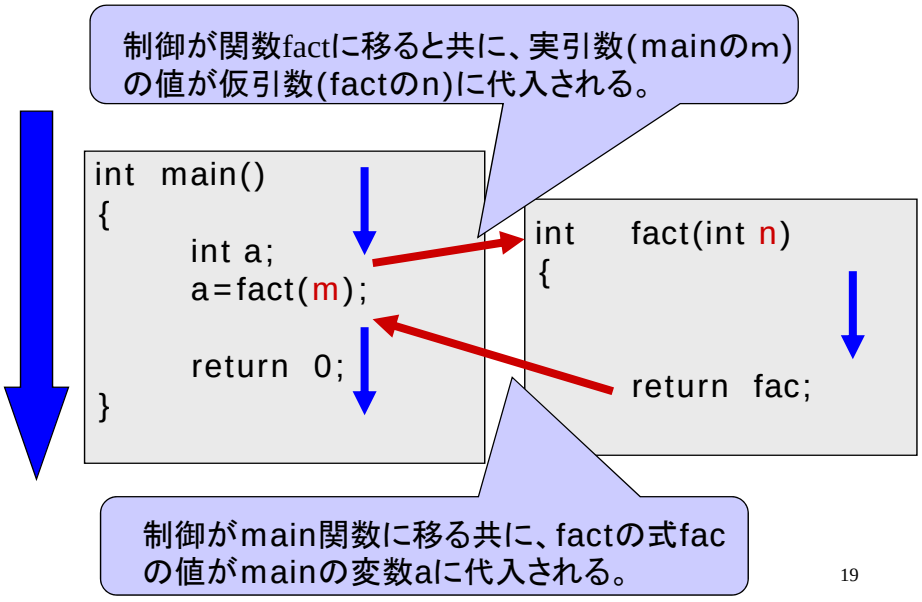
```
int     fact(int n)
{
        return  fac;
}
```



(a=facという代入動作が
行われる。)

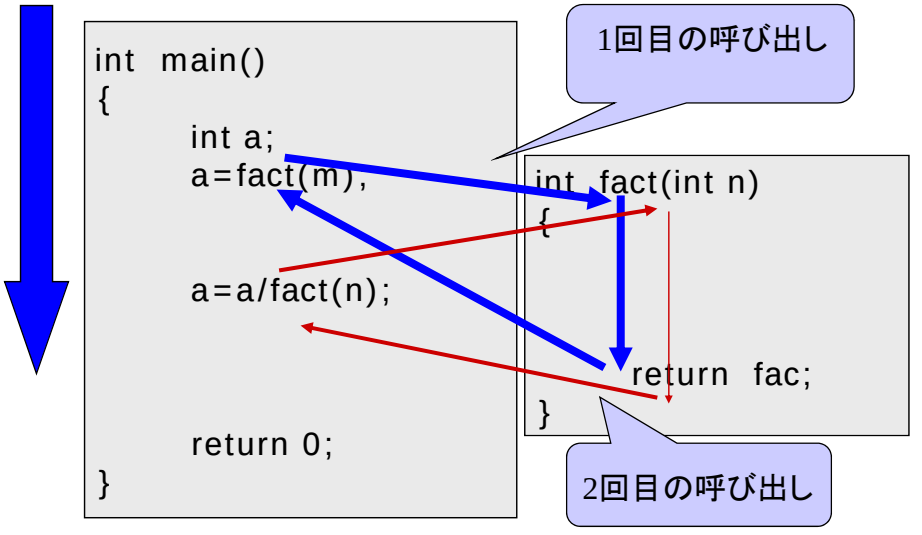
18

関数が複数あるときの制御の流れ1



19

関数が複数あるときの制御の流れ2

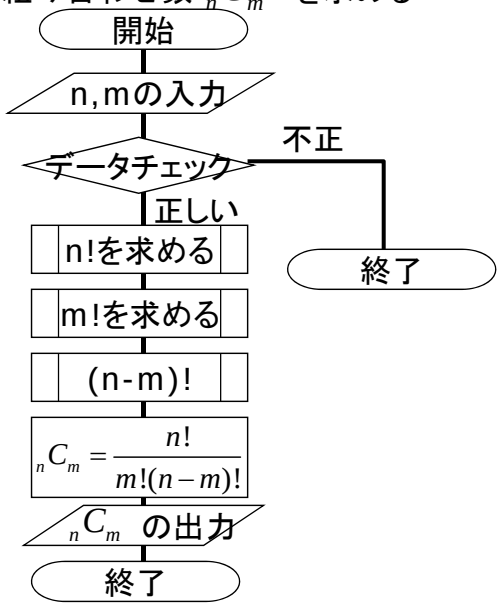


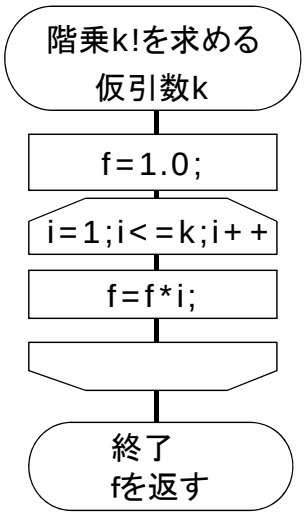
同じような処理を複数回行いたいときにも、
関数を用いると便利。

20

関数を表わすフローチャート

組み合わせ数 ${}_nC_m$ を求める





練習1

```
/*test_fanction.c　関数実験1　コメント省略*/
#include <stdio.h>
int  switch_sign(int);
int  main()
{       int a;
        int b;
        printf(”整数を入力して下さい。a= ?");
        scanf("%d",&a);
        printf(“関数呼び出し前　a=%d : b= %d ¥n”,a,b);
        b=switch_sign(a);
        printf(“関数呼び出し後　a=%d : b= %d ¥n”,a,b);
        printf("%d  = switch_sign( %d) ¥n",b,a);
        return  0;
}/*   つづく*/
```

```
int  switch_sign(int    c)
{
        printf(“関数switch_sign実行中¥n”);
        printf(“c=%d¥n”,c);
        return    -c;
}
```

23

voidという型

return;

という文を持つ関数には戻り値がない。
このように、戻り値がないことをvoidという型であらわす。

```
void fanc1()
{
        return ;
}
```

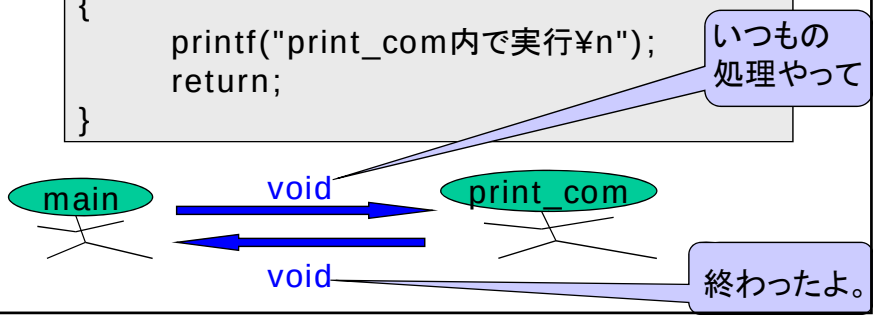
あるいは、仮引数がないことを
明示的にvoidという型で表す。

```
void fanc2(void)
{
        return ;
}
```

24

練習2

```
/*test_void.c  関数実験2  コメント省略*/
#include<stdio.h>
void print_com(void);
int main()
{
        print_com( );
        return  0;
}

void print_com(void)
{
        printf("print_com内で実行¥n");
        return;
}
```



複数の関数を持つプログラムの書き方

書式

```
 /*          プロトタイプ宣言       */
型1 関数名1(型1a  仮引数1a, 型1b 仮引数1b, ･･･);
型2  関数名2(型2a 仮引数2a, 型2b 仮引数2b, ･･･);
int main()
{
}

型1 関数名1(型1a  仮引数1a, 型1b  仮引数1b, ･･･)
{
}
型2  関数名2(型2a  仮引数2a, 型2b  仮引数2b,･･･)
{
}
```

練習3

```
/*test_fanction3.c　関数実験3　コメント省略*/
/*ヘッダファイルの取り込み*/
#include <stdio.h>

/*プロトタイプ宣言*/
void　print_com(void); /*　コメントを表示する関数　*/
int　plus(int,int);　　/*　和を求める関数 */
int　main()
{
　　int a;
　　int b;
　　int c;

/*次に続く */
```

27

```
/*続き*/
　　printf(”整数を入力して下さい。a= ?");
　　scanf("%d",&a);
　　printf(”整数を入力して下さい。b= ?");
　　scanf("%d",&b);

　　/*引数と戻り値が無い関数呼び出し*/
　　print_com( );

　　/*引数を基に与えて戻り値を受け取る関数
　　呼び出し。*/
　　c=plus(a,b);


　　printf("%d　= plus( %d,%d) ¥n",c,b,a);
　　return　0;
}
```

28

```
/*続き*/
/*コメントを表示する関数
        仮引数:なし(void)
        戻り値:なし(void)*/
void   print_com(void)
{
        printf("計算開始¥n");
        return;
}
/*print_com終了*/
```

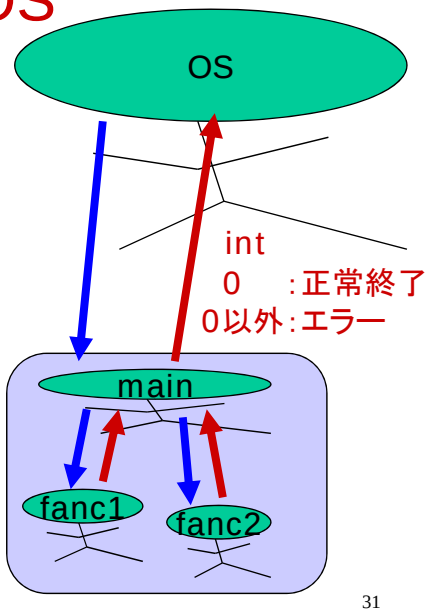
この関数のように、
値のやりとりがなくても、
なにか動作することがあります。
これを副作用といいます。

```
/*続き*/
/*2つの整数の和を計算する関数
        仮引数x:被演算項1(整数)
        仮引数y:被演算項2(整数)
        戻り値:"被演算項1＋被演算項2"の値を返す。*/
int     plus(int    x, int  y)
{
        /*変数宣言*/
        int   z;

        /*計算*/
        z=x+y;
        return  z;
}
/*plus終了*/
/*全プログラム(test_fanction3.c)終了*/
```

こんな風に、
関数にコメントを付けること。
(スタイル規則参照。)

関数mainの型とOS

```
/*  aaaaa.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。

31

変数のスコープ1（有効範囲1）

関数定義の一般的な書式:

```
型1  関数名1(型1a  仮引数1a, 型1b 仮引数1b, ･･･)
{
        /*変数宣言*/
        型x      変数x

}
```

引数が複数ある場合は、
引数リスト中で
各引数をカンマ「, 」
で区切る。

仮引数とその関数内で宣言した変数は、
宣言した関数の内部だけで有効である。

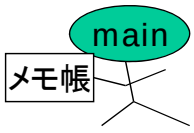
したがって、異なる2つの関数で同じ変数名を用いても、
それぞれの関数内で別々の変数としてあつかわれる。

32

```
int     main()
{
        /*変数宣言 */
        mainの変数

}
型1     関数1(型1a  仮引数1a,型1b 仮引数1b)
{
        /*変数宣言*/
        関数1の変数

}
型2     関数2(型2a  仮引数2a,型2b 仮引数2b)
{
        /*変数宣言*/
        関数2の変数

}
```

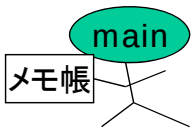

イメージ

いままでは、main関数1つしかなかった。



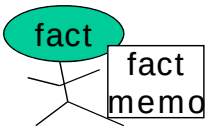
1人で仕事をする。
自分用の変数をつかう。

mainとfactがあると



お願い

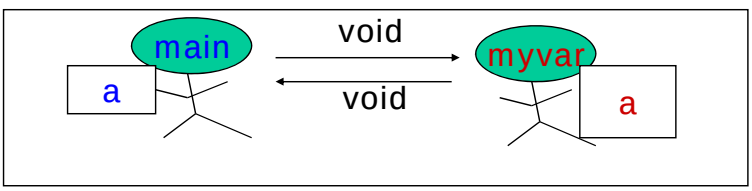
このデータ
で仕事して



練習4

```
/*test_scope.c  関数実験4  コメント省略*/
#include<stdio.h>
void  myvar(void);
int main()
{
      int a;

      printf("( In main)  Input a= ? ");
      scanf("%d",&a);
      printf("(In main) a= %d ¥n",a);
      myvar();
      printf("(In main) a= %d ¥n",a);
      return  0;
}
/*  次に続く  */
```

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

      printf("( In myvar)  Input a= ? ");
      scanf("%d",&a);
      printf("(In myvar) a= %d ¥n",a);

      return;
}
```



グローバル変数とローカル変数

実は、関数の外でも、変数の宣言ができます。
その変数をグローバル変数と呼びます。

一般的な形

```
/*グローバル変数宣言*/
型A    変数A;
型B    変数B;

int main()
{
        /*mainのローカル変数宣言*/
}
```

本演習では、
グローバル変数は、
1文字だけ英大文字
で残り小文字にしましょう。
（スタイル規則参照）

```
int   Global1;
```

37

グローバル変数のスコープ

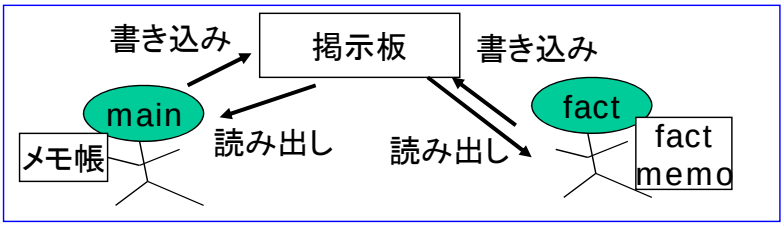
```
/*グローバル変数宣言*/
グローバル変数

int     main()
{
        /*変数宣言 */
        mainの変数
}
型1    関数1(型1a  仮引数1a,型1b 仮引数1b)
{
        /*変数宣言*/
        関数1の変数
}
```

38

ローカル変数とグローバル変数

ローカル変数は、自分用のメモ帳

グローバル変数は、どの関数でも読み書きできる掲示版



39

グローバル変数とローカル変数が同じ名前のときは？

ある関数でグローバル変数と同じ名前の変数を宣言すると、
その関数内ではその変数名はローカル変数としてあつかわれる。
したがって、グローバル変数の変更はおこなわれない。
（変数名が同じもの同士では、
そのスコープが狭いものが優先される。
関数が違えば、同じ変数名でも大丈夫。）

注意：
本演習のスタイルでは、
ローカル変数とグローバル変数は必ず異なる。
ローカル変数：すべて小文字
グローバル変数：1文字目大文字
マクロ名：すべて大文字

40

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:B0zB0xx
        ソースファイル:combi1.c
        実行ファイル:combi1
        説明:組み合わせ数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数nCmを出力。
*/
#include <stdio.h>

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

```
/*      続き    */
/*main関数*/
int     main()
{
        /*ローカル変数宣言*/
        int     n;      /*nCmのn*/
        int     m;      /*nCmのm*/
        int     com;            /*組み合わせ数nCm*/

/*      次のページに続く      */
```

```
    /*    続き   */
    /*    入力処理 */
    printf("組み合わせ数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。*/
    /*    次ページへ続く       */
```

```
    /*    続き   */

    /*    組み合わせ数を計算 */
    com=fact(n)/(fact(m)*fact(n-m));

    /*出力処理*/
    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: n!のn
                (0以上15未満の値とする。)
        戻り値:n!を返す。*/
int     fact(int n)
{
        /*      ローカル変数宣言    */
        int     i;      /*ループカウンタ*/
        int     fac;    /*階乗n!*/

        fac=1;          /*0！＝1であるので1を代入*/

        /*      次に続く        */
```

45

```
        /*      続き */
        /*計算処理*/
        for(i=1;i<=n;i++)
           {
              /*階乗の計算*/
              fac=fac*i;
           }

        return fac;  /*facの値n!を戻す*/
}
/*factの終了*/
/*全てのプログラム(combi1.c)の終了*/
```

46

実行例

```
$make
gcc  combi1.c  -o combi1
$ ./combi1
組み合わせ数nCm を計算します。
Input  n=? 4
Input  m=? 3
4C3  = 4
$
```

```
$./combi1
組み合わせ数nCm を計算します。
Input  n=? 4
Input m=? 5
不正な入力です。
$
```