

# Chapter 7 Functions

7.1 Foundation of Functions

7.2 Nested Calling and Recursive Calling

7.3 Arrays Qua Parameters of Functions

7.4 The Scope of Variables

7.5 The Storage Type of Variables

7.6 Local Functions and External Functions

# Foreward

- [ex.7.1] Compute the Combination  $c_m^n = \frac{m!}{n!(m-n)!}$

```
main()
{ int m,n,i;
  long fact,c;
  scanf("%d%d",&m,&n);
  fact=1;
  for(i=2;i<=m;i++)
    fact*=i;
  c=fact;
```

```
fact=1;
for(i=2;i<=n;i++)
  fact*=i;
```

```
c=c/fact;
```

```
fact=1;
for(i=2;i<=m-n;i++)
  fact*=i;
```

```
c=c/fact;
```

```
printf("c=%d\n",c);
```

```
}
```

## Design a function

```
/* define a  
function fact */  
long fact(int k)  
{ int i;  
  long y=1;  
  for(i=1;i<=k;i++)  
    y*=i;  
  return(y);  
}
```

formal parameter  
(形参) k

/\*fact is called by main()\*/

```
main()  
{ int m,n;  
  long c;  
  scanf("%d%d",&m,&n);  
  c=fact(m); /*calling fact*/  
  c=c/fact(n);  
  c=c/fact(m-n);  
  printf("c=%d\n",c);  
}  
c=fact(m)/(fact(n)*fact(m-n));
```

actual parameters (实参) m,n,m-n

## **Basic concept**

**function declarations; formal parameters (形式参数) ;the value the function returns; the result-type of the function; function call; actual parameters; parameter passing**

# 7.1 Foundation of Functions

- **DECLARATIONS OF FUNCTIONS**

```
/*function fact */  
long fact(int k)  
{  
    int i;  
    long y=1;  
    for(i=1;i<=k;i++)  
        y*=i;  
    return(y);  
}
```

## 1. Five factors of functions

**Function name**

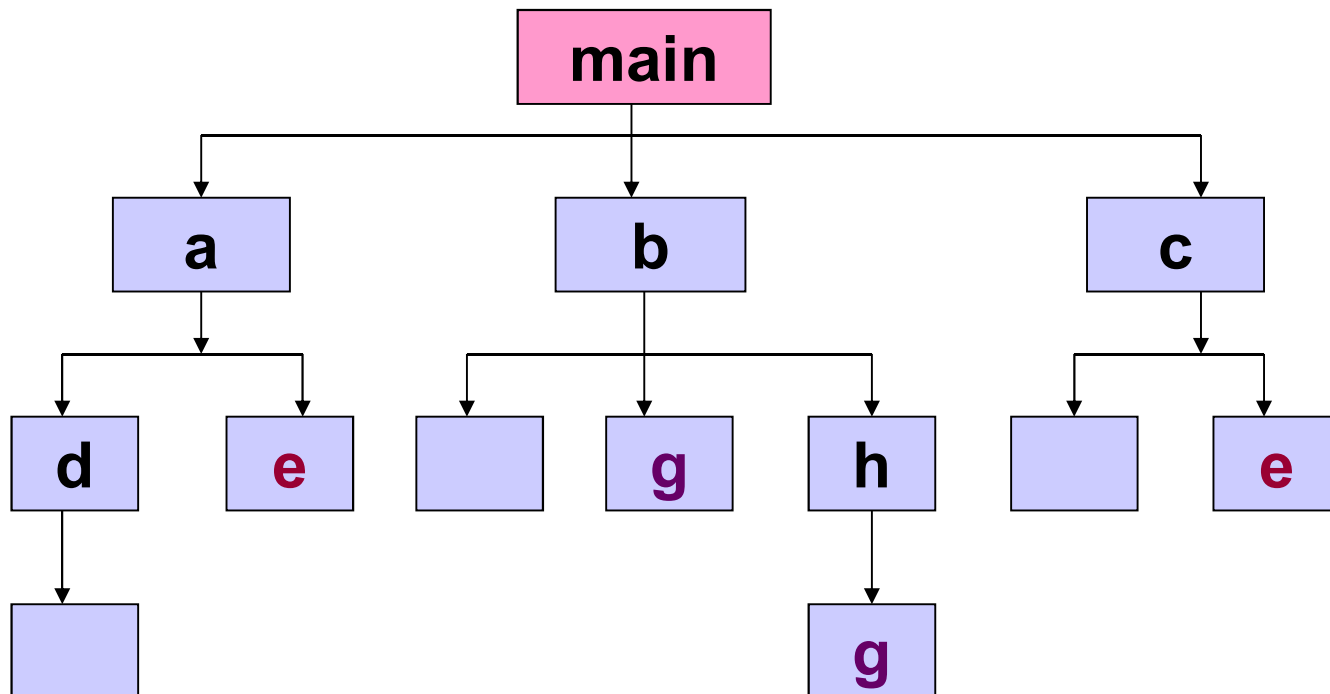
**Declaration of arguments(  
name and type of arguments)**

**Type of function value**

**Function**

**Returning value of function**

2. A program is just a set of definitions of variables and functions.



Each function definition has the form

```
result-type function-name(argument-specifiers)
{
    declarations and statements
}
```

Various parts may be absent; a minimal function is

```
dummy() {}
```

# Sorts of Functions

- by use

library functions

functions defined by user

- by format

functions take arguments or don't take arguments

null functions

functions return result or don not return result



**Give following functions' type**

```
void printline(void)  
{ int i;  
  for(i=1;i<=30;i++)  
    printf("-");  
  printf("\n");  
}
```

```
long fact(int k)  
{ int i;  
  long y=1;  
  for(i=1;i<=k;i++)  
    y*=i;  
  return(y);  
}
```

```
main()  
{ int m,n;  
  long c;  
  clrscr();  
  printline();  
  printf("m,n=");  
  scanf("%d%d",&m,&n);  
  c=fact(m)/(fact(n)*fact(m-n));  
  printline();  
  printf("c=%d\n",c);  
  getch();  
}
```

[ex.7.2] Define a function to find the max one of two numbers

main()

```
{ int max(int n1, int n2);    /*declaration*/  
  int num1,num2;  
  printf("input two numbers:\n");  
  scanf("%d%d", &num1, &num2);  
  printf("max=%d\n", max( num1 , num2 ));  
  getch();  
}
```

parameter passing

```
int max( int n1, int n2 )    /*define max()*/  
{ return ( n1>n2?n1:n2 );  
}
```

[ex.7.3] output all of primes between 2 and 200

```
main()
{ int i, j, counter=0;
  for(i=2 ; i<=200; i++)
  { for(j=2; j<=i-1; j++)
    if(i%j==0)
      break;
    if( j == i )
      { printf("%8d",i);
        counter++;
      }
  }
}
```

```
printf("\ncounter=%d\n",counter);
}
```

```
main()
{ int i, j, counter=0;
  for(i=2 ; i<=200; i++)
  { if (prime( i ))
    { printf("%8d",i);
      counter++;
    }
  }
  printf("\ncounter=");
  printf("%d\n",counter);
}
```

Design a function

```
int prime(int k)
{ int i, p;
  for(i=2; i<=k-1; i++)
    if( k%i==0)
      break;
```

```
    if( i==k )
      p=1;
    else
      p=0;
    return(p);
```

```
}
```

```
main()
{ int i, j, counter=0;
  for(i=2 ; i<=200; i++)
    { if ( prime( i ) )
      { printf("%8d",i);
        counter++;
      }
    }
  printf("\ncounter=");
  printf("%d\n",counter);
}
```

**return(i==k);**

## 7.1.3 Declaration of Called Function and prototype of Function

- When the called function is a custom function and is in the same program file as the keynote function, the function is declared in the keynote function.

Declaration form

the head of function+semicolon

When function declaration can be left out

(1)definition of the called function before the calling function;

(2)type of function and type of parameters are all int

## 7.1.4 Call a Function

Calling form:

function name( [actual Parameters list] )

Calling mode:

(1) In expression

ex.     ave=(sum(a)-max(a)-min(a))/n;

          m=max(n3, max(n1,n2) );

          printf(“%d”,max(n1,n2));

(2) Function statement   (function type is void)

ex.     println();

## 7.1.5 Formal Parameters and Actual Parameters

Function parameters {  
    Formal Parameters  
    Actual Parameters

Formal Parameters appear in function definition, only be use in function body.

Transfer data between parameters:

when a function is called, the calling function transfer actual parameters' value to formal parameters of the called function unilaterally (单方面)

## **[ex. 7.4 ] Data transmission between actual parameters and formal parameters**

```
void main()
```

```
    { void s(int n);  
      int n=100;  
      s(n);  
      printf("n_s=%d\n",n);  
      getch();  
    }
```


```
void s(int n)
```

```
    { int i;  
      printf("n_x=%d\n",n);  
      for(i=n-1; i>=1; i--) n =n+i;  
      printf("n_x=%d\n",n);  
    }
```



```
void main()
{ void s(int n);
  int n=100;
  s(n);
  printf("n_s=%d\n",n);
  getch();
}

void s(int n)
{ int i;
  printf("n_x=%d\n",n);
  for(i=n-1; i>=1; i--) n =n+i;
  printf("n_x=%d\n",n);
}
```



Actual parameter: **n**

Formal parameters : **n**

**result:**

**n\_x=100**

**n\_x=5050**

**n\_s=100**

## 7.16 Function Returning Non-value

In C, a function that does not return a value is identified by using the keyword *void* in the function prototype in place of the result type. For example, the prototype

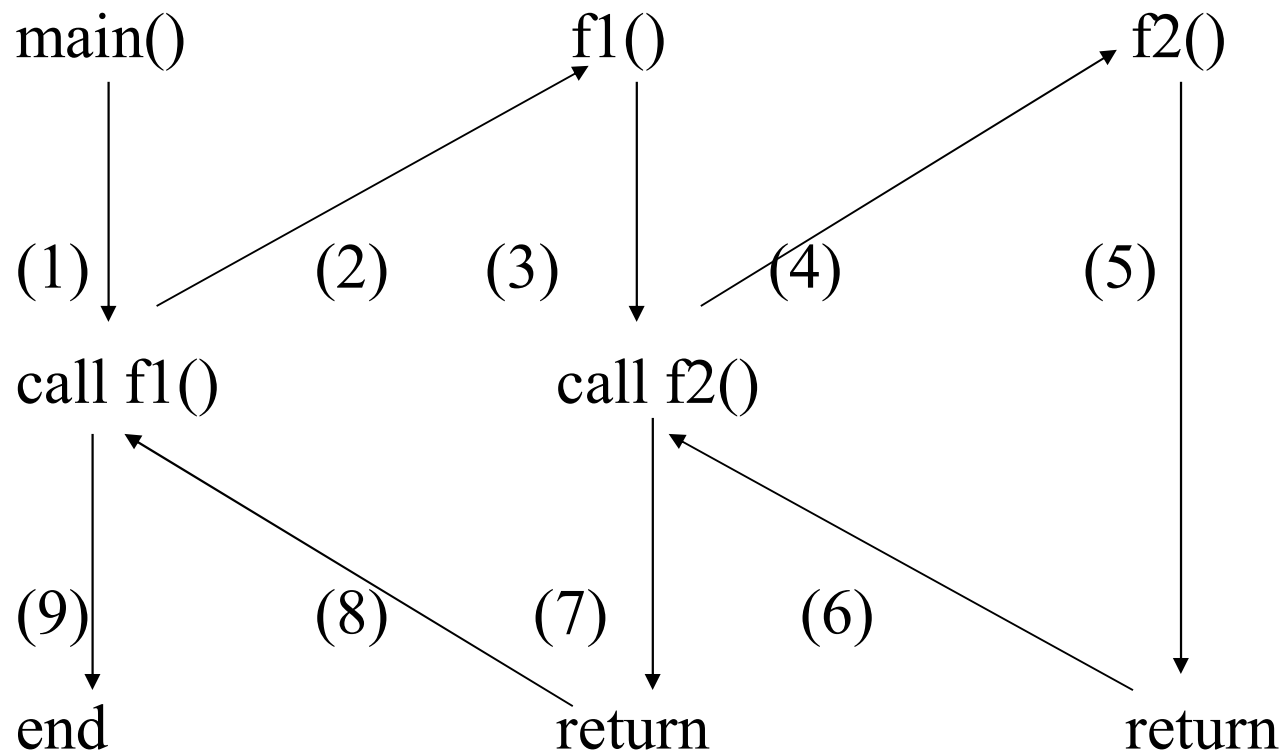
```
void printline( );
```

**Declares a function that takes no arguments and returns no result.**

## 7.2 Nested Calling and Recursive Calling

### •7.2.1 Nested Calling

For example:



[ex.7.5] calculate  $s=1^k+2^k+3^k+\dots+N^k$

```
long f1(int n, int k)
```

```
{    long sum=0;
```

```
    int i;
```

```
    for(i=1;i<=n;i++) sum += f2(i, k);
```

```
    return(sum);
```

```
}
```

```
long f2(int n, int k)
```

```
{    long power =n;
```

```
    int i;
```

```
    for(i=1;i<k;i++) power *= n;
```

```
    return( power);
```

```
}
```

```
main()
```

```
{ printf("n,k=");scanf("%d%d",n,k);  
    printf("Sum of %d powers of integers from 1 to  
        %d = ",k,n);  
    printf("%d\n",f1(n,k));  
    getch();  
}
```

## 7.2.2 Recursive Calling

Recursive Calling, A function calls itself.

- Recursion is divided into: direct recursion, indirect recursion.
- Recursion can simplify complex processes, however, it could cost more time and taking up more memory space.
- Recursive conditions:
  1. problem decomposition (large to small)
  - 2 . Small to a certain extent can be directly solved (with if implementation)

[ex. 7.6 ] calculate age

$$\text{age}(5)=\text{age}(4)+2$$


$$\text{age}(4)=\text{age}(3)+2$$



$$\text{age}(3)=\text{age}(2)+2$$


$$\text{age}(2)=\text{age}(1)+2$$

$$\text{age}(1)=10$$


$$\text{age}(n)=\begin{cases} 10 & (n=1) \\ \text{age}(n-1)+2 & (n>1) \end{cases}$$



$$\begin{array}{l} \text{age}(5) \\ = \text{age}(4) + 2 \end{array}$$

$$\begin{array}{l} \text{age}(4) \\ = \text{age}(3) + 2 \end{array}$$


$$\begin{array}{l} \text{age}(3) \\ = \text{age}(2) + 2 \end{array}$$


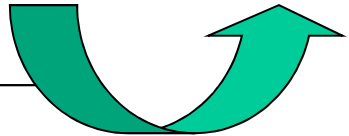
$$\begin{array}{l} \text{age}(2) \\ = \text{age}(1) + 2 \end{array}$$


$$\begin{array}{l} \text{age}(1) \\ = 10 \end{array}$$

$$\begin{array}{l} \text{age}(5) \\ = 18 \end{array}$$

$$\begin{array}{l} \text{age}(4) \\ = 16 \end{array}$$


$$\begin{array}{l} \text{age}(3) \\ = 14 \end{array}$$


$$\begin{array}{l} \text{age}(2) \\ = 12 \end{array}$$


$$\begin{array}{l} \text{age}(1) \\ = 10 \end{array}$$



```

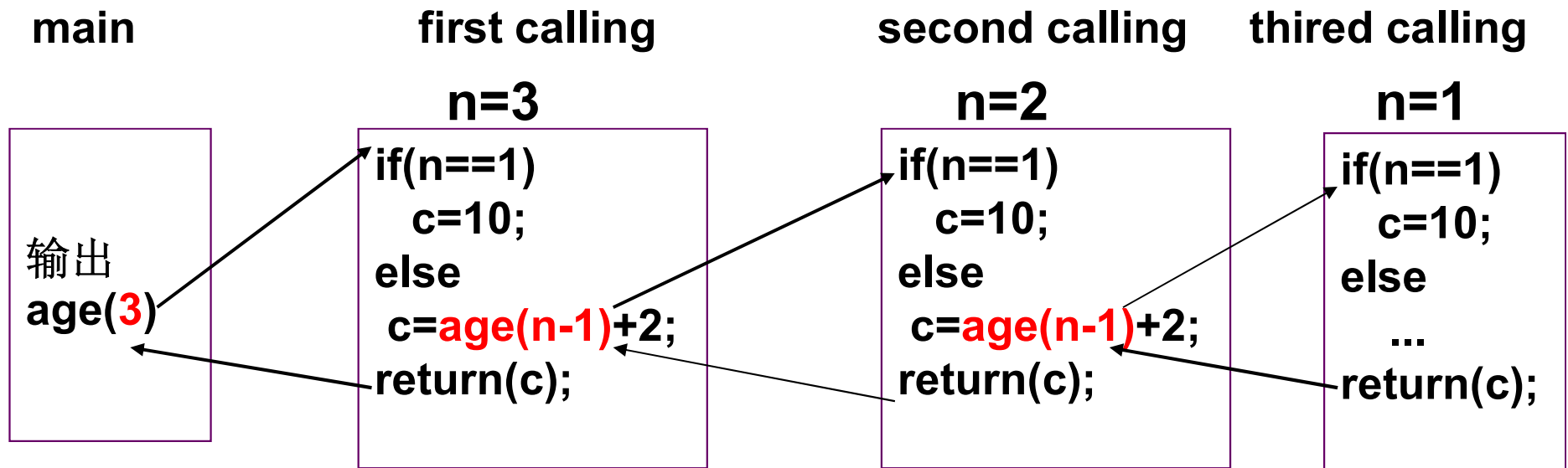
age( int n)
{
    int c;
    if(n==1) c=10;
    else c=age(n-1)+2;
    return(c);
}

```

```

main()
{
    printf("%d",age(5));
}

```



[ex.7.7] calculate  $n!$

$$n! = \begin{cases} n \cdot (n-1)! & n > 1 \\ 1 & n = 1 \end{cases}$$

```
long power(int n)
```

```
{ long f;  
  if(n>1)  
    f=power(n-1)*n;  
  else  
    f=1;  
  return(f);  
}
```

```
main()
```

```
long power( int n );
```

```
{ int n;  
  long y;  
  printf("n=");   scanf("%d",&n);  
  y=power(n);  
  printf("%d!=%ld\n",n,y);  
}
```

```
long power(int n)
```

```
{ long f;  
  if(n>1) f=power(n-1)*n;  
  else f=1;  
  return(f); }
```

Recursive Calling !

**【ex. 7.8 】 The expression is**

$$f(x, n) = \sqrt{n + \sqrt{(n-1) + \sqrt{(n-2) + \sqrt{\cdots + \sqrt{1 + x}}}}}$$

calculate the value of  $f, x=3.1, n=15$  及  $x=8.1, n=10$ .

**recursion formula**

$$f(x, n) = \begin{cases} \sqrt{1 + x} & , n=1 \\ \sqrt{n + f(x, n-1)} & , n>1 \end{cases}$$

```
#include <math.h>
```

```
main()
```

```
{ float f(float x,int n);  
  printf("f(3.1,15)=%f\n",f(3.1,15));  
  printf("f(8.1,10)=%f\n",f(8.1,10));  
}
```

```
float f(float x, int n)
```

```
{ float y;  
  if(n==1) y=sqrt(1+x);  
  else y=sqrt(n+f(x,n-1));  
  return(y);  
}
```

# Summary of Recursion

## 一、Recursive conditions:

1 problems can be decomposed

The larger, more difficult to solve the big problem of small, easy to solve the same problem

2 small to a certain extent can be directly solved

## 二、Advantages of recursion:

Make complex problems simple, short and clear

## 三、Disadvantages of recursion:

Long running time, occupy more storage space.

四、Not every problem is suitable to be solved by recursive method.