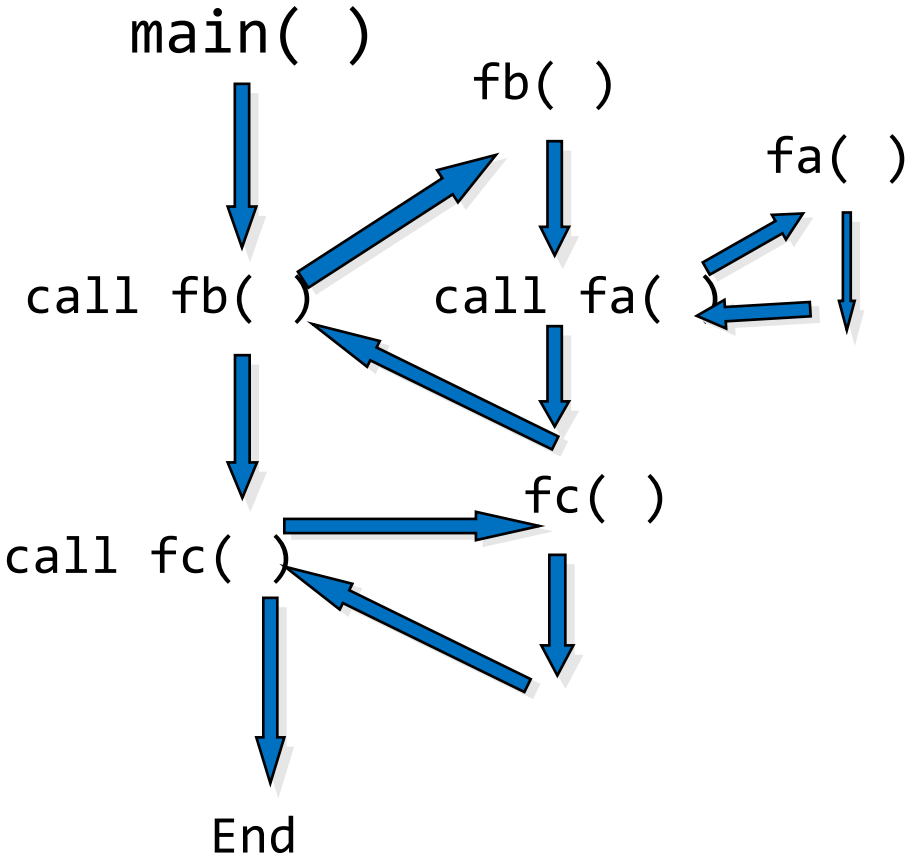
Class 8 Function(函数)-2

Nesting of Functions (函数嵌套)

- C permits nesting of functions freely.
  - *main* can call *function1*, which calls *function2*, which call *fucntion3*
- The main function cannot be called by any other functions including itself.
- Nesting calls are permitted, but no nesting definitions!

Examples of Nested Functions

```
void fa( ){...};
int fb( ){...fa();...};
int fc( ){...};

int main( )
{
    fb( );
    fc( );
    return 0;
}
```



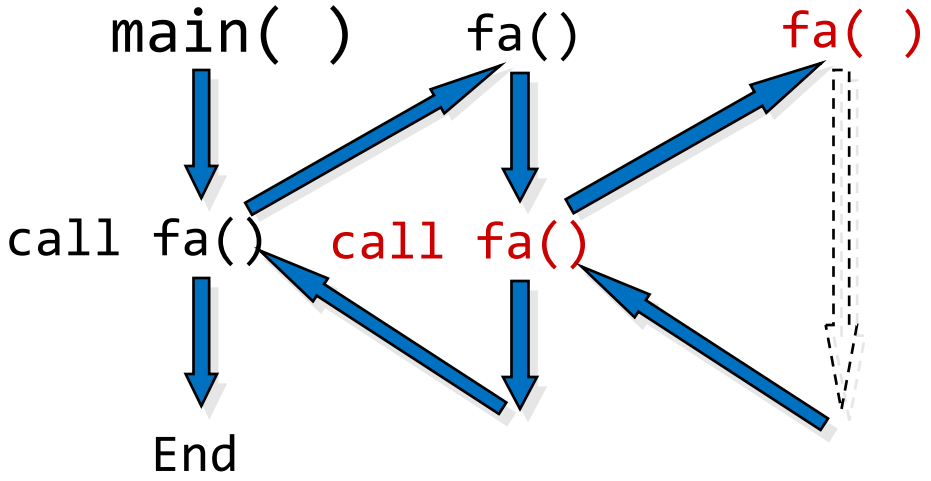
Function Recursion (函数递归)

- Recursion can be regarded as a special case of nested process, where a function calls itself.

- Two key issues to design a recursion
  - Divide action (convert to a smaller self-similar problem)
  - End condition (jump out from recursion)

Function Recursion

```
void fa( )
{
    ...
    fa( );
    ...
}
int main( )
{
    fa( );
    return 0;
}
```



A classical example

- The factorial number n (n的阶乘) is expressed as a series of repetitive multiplications as shown below:

  factorial of n = n(n-1)(n-2)……1.

- New formula (formulation)

$$\text{fact}(n) = n! = \begin{cases} 1 & if \quad n = 0 \\ n \times (n-1)! & if \quad n \geq 1 \end{cases}$$

- Convert to pseudo-codes

```
Divide:
      if (n > 0) n!=n*(n-1)!
End of recursion :
     if (n equals 0) n!=1
```

Program

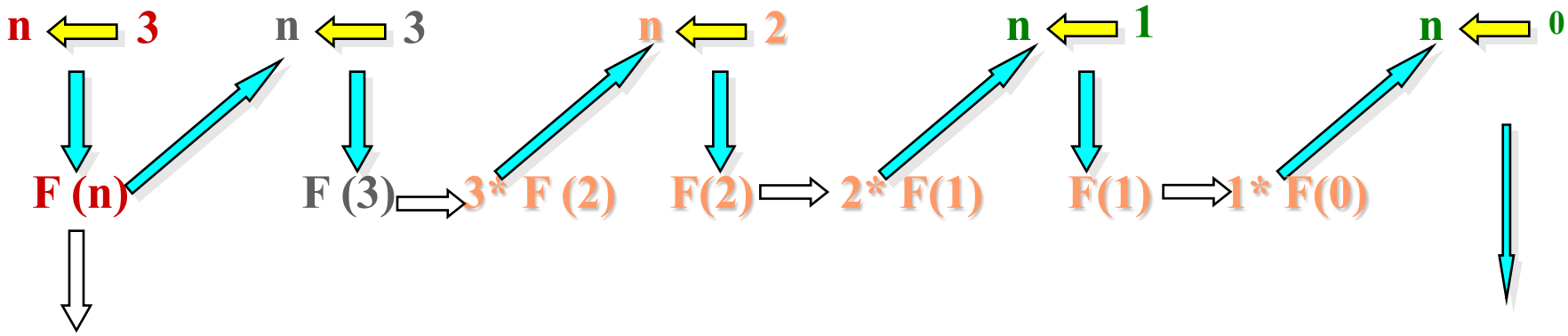
```
int Fact(int n)
{
    if(n>0)
       /*call face recursively*/
       return  n*Fact(n-1);
    else
       return 1;
}
```

Divide Action
(call itself)

End
Recursion

```
int Fact(int n)
{
    if(n>0)
       /*call face recursively*/
       return  n*Fact(n-1);
    else
       return 1;
}
```

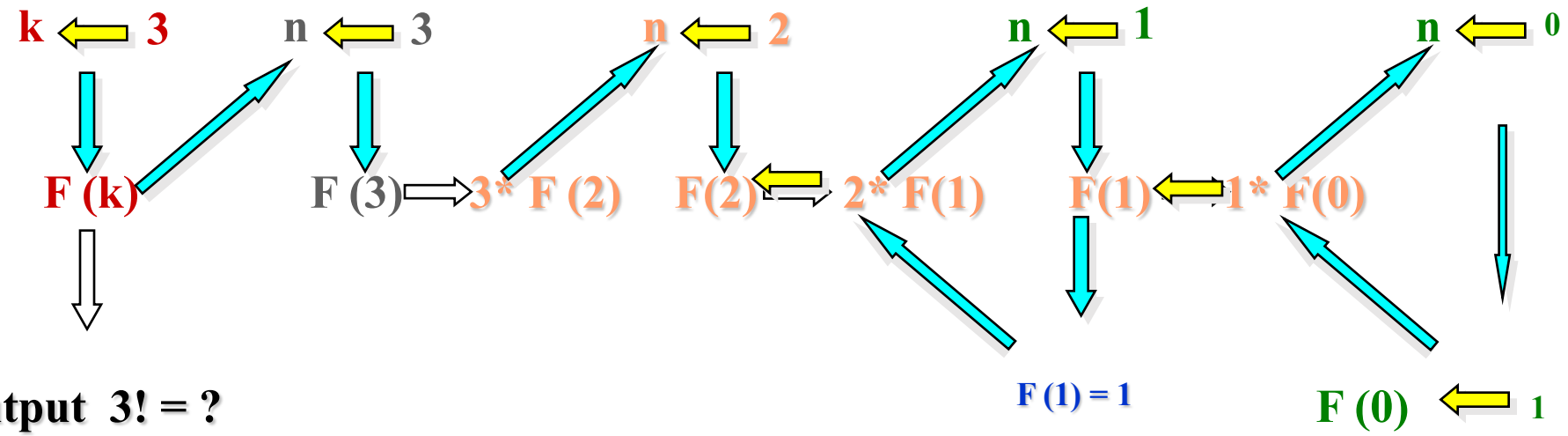
Fact (3) = 3!



Output 3! = ?

```
int Fact(int n)
{
    if(n>0)
       /*call face recursively*/
       return  n*Fact(n-1);
    else
       return 1;
}
```

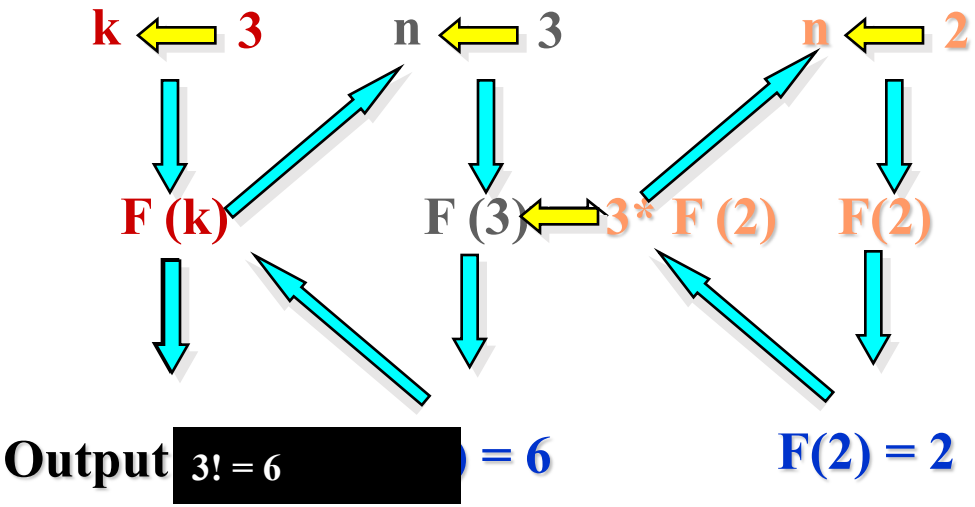
Fact (3) = 3!



Output 3! = ?

```
int Fact(int n)
{
    if(n>0)
       /*call face recursively*/
       return  n*Fact(n-1);
    else
       return 1;
}
```

Fact (3) = 3!



Example

- Write a function to calculate a serious of factorial numbers (1! 2! 3! … n!). Use an array name as the formal parameter.

```
int calNumbers(int nums[], int n)
{
    ...
}
```

```
#include <stdio.h>
#define SIZE 10

int Fact(int n)
{
    if ( n > 0 )
        return n*Fact(n-1);
    else
        return 1;
}
int calNumbers(int nums[], int n)
{
    int i;
    if ( n > 0 && n < SIZE+1 )
    {
        for ( i = 0 ; i < n ; i++ )
            nums[i] = Fact(i+1);
        return 1;
    }
    else
        return 0;
}
...
```

```
int main()
{
    int nums[SIZE];
    int i, n;
    scanf("%d", &n);
    if ( !calNumbers(nums, n) )
    {
        printf(
          “Illegal Number\n");
        return 1;
    }
    for ( i = 0 ; i < n ; i++ )
        printf("%d ", nums[i]);
    printf("\n");

    return 0;
}
```

Exercise

- Using recursion to write a function to sum all items in an array A = {a_1, a_2, …, a_n}

$$sum(n) = \sum_{i-1}^{n} a_n$$

- Rewrite the equation

$$sum(n) = \begin{cases} a_n + \sum_{i=1}^{n-1} a_i & n > 1 \\ a_1 & n = 1 \end{cases}$$

***sum*(*n*-1)**

```
int sum(int nums[], int n)
```

```c
#include <stdio.h>
#define SIZE 10

int sum(int nums[], int n)
{
    if ( n > 1 )
        return nums[n-1]+sum(nums, n-1);
    else
        return nums[0];
}

int main()
{
    int nums[SIZE];
    int i;
    for ( i = 0 ; i < SIZE ; i++ )
        scanf("%d", &nums[i]);

    printf("sum = %d\n", sum(nums ,SIZE));
    return 0;
}
```

Scope and Life Time for Variables

- Each variable is stored in memory in C language, and different variables have different scope and life time.

- Scope (作用域)
  - The region of a program in which a variable (name) is available for use.

- Life Time (生存期)
  - the duration of time in which a variable exists in the memory during execution

Scope of Variables

- Local Variable (局部变量)
  - The variable that is defined inside functions or blocks
- Global Variable (全局变量)
  - The variable that is defined outside functions

```
int a, b; //global variables


void funs()
{
    int c, d; //local variables
    ...
}

int main()
{
    int i, j; //local variables
    ...
}
```

Local Variables

- Local variables are also called as internal variables (内部变量)
- Local variables are only available inside the block or function where they are defined.
- Memory of Local variables are released automatically when the function is finished

```
int max(int iA, int iB)
{
    int iC;
    if (iA > iB)  return iA;
    else    return iB;
}

int add(int iD, int iE)
{
    int iK;
    iK = iD + iE;
    return iK;
}

int main()
{
    int iM = 1; iN = 2;
    printf("%d %d\n", max(iM, iN),
           add(iM. iN));
    return 0;
}
```

Notes of Local Variables

- Variables defined inside a main function can ONLY be used inside the main function, CAN NOT be used in the other functions.

- Formal variables are the local variables for the called function, and the actual variable are local variables for the main function.

- In different functions, the names of local variables can be the same, but they are different variables.

```
void func1() {int i; …}

void func2() {int i; …}
```

Local Variables inside a Block

- Local variables that are defined inside a block can be only used inside that block
- IF two variables that is defined inside and outside a block have the same names, only the inside variable is available inside the block

```
#include <stdio.h>

int main()
{
    int iJ = 1, iK = 2;
    {
        int iK = 3, iL = 4;

        printf("%d\n", iJ); // 1
        printf("%d\n", iK); // 3
        printf("%d\n", iL); // 4
    }

    printf("%d\n", iJ); // 1
    printf("%d\n", iK); // 2
    printf("%d\n", iL); // error

    return 0;
}
```

Global Variables

- Global variable is defined outside functions, and also called external variables
- Global variable does not belong to any functions, and can be used after the place where it is defined
- Global variable can be used before its definition if it is declared by using “extern” keyword

```
int iA, iB; //global variables

void f1()
{
   //fX, fY can be used here
   extern float fX, fY;
   ...
}

float fX, fY; //global variables

int f2()
{
   ...
}

int main()
{
   ...
}
```

By using the global variable, the function can change several values.

```
#include <stdio.h>

int iArea; // global variable

int volumeAndArea (int iL,int iW,int iH)
{
    iArea=2*(iL*iW+iL*iH+iW*iH);//save to global variable
    return iL*iW*iH;
}
int main()
{
    int iL,iW,iH,iVolume;
    printf("input length,width and height\n");
    scanf("%d%d%d",&iL,&iW,&iH);
    iVolume=volumeAndArea(iL,iW,iH);
    printf("\nvolume=%d\narea=%d\n",iVolume,iArea);
    return 0;
}
```

If the names of global and local variables are the same, the global variable is not available inside the function

```
#include <stdio.h>

int iArea; // global variable

int volumeAndArea (int iL,int iW,int iH)
{
    int iArea; // local variable
    iArea=2*(iL*iW+iL*iH+iW*iH); //local variable is changed
    return iL*iW*iH;
}
int main()
{
    int iL,iW,iH,iVolume;
    printf("input length,width and height\n");
    scanf("%d%d%d",&iL,&iW,&iH);
    iVolume=volumeAndArea(iL,iW,iH);
    printf("\nvolume=%d\narea=%d\n",iVolume,iArea);
    return 0;
}
```

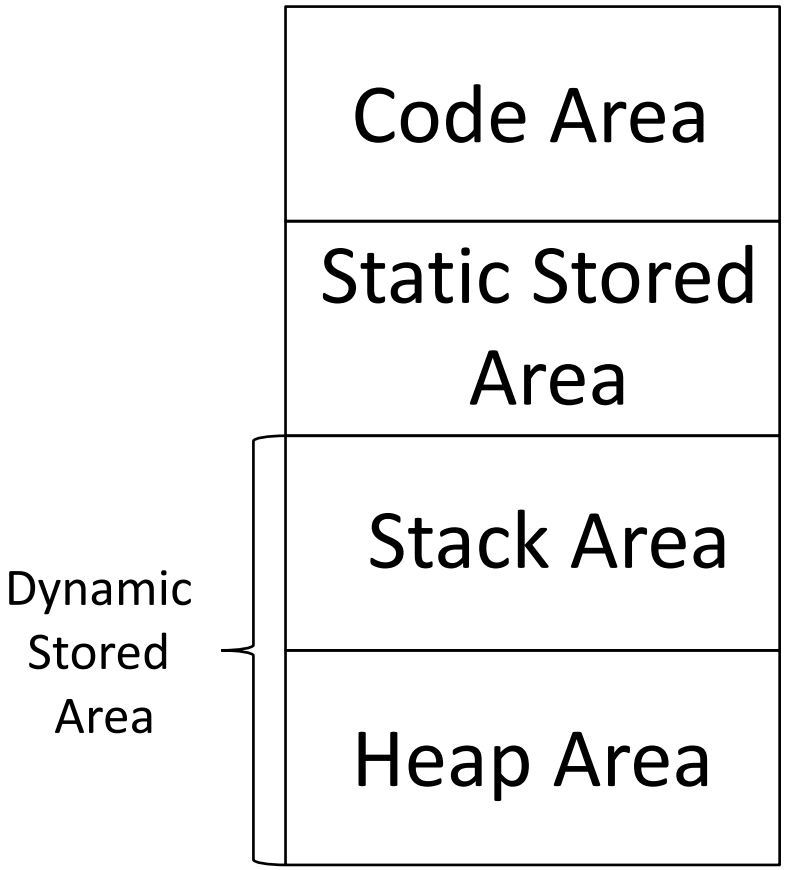
Notes of Global Variables

- The names of global variables and local variables can be the same, but inside the scope of the local variable the global variable can not be available

- Only use global variable when it is necessary

- The global variables take the static stored area of memory (静态存储区), and the local variables take the stack area (栈区) of memory

Summary of Memory in C Program



Codes of Functions

Constant, Global Variable, Static Variables

Local Variables, Formal Parameters, Returned Values

Memory allocated by malloc()

Storage of Variables

- Variables for Different Kinds of Storage
  - Auto Variable (自动变量)
  - Register Variable (寄存器变量)
  - Extern Variable (外部变量)
  - Static Variable (静态变量)

Auto Variable (自动变量)

- The most used variable in C Program
  - In default, all variables are auto variables
  - Usually auto is omitted
  - Another name of local variables
- Storage
  - It is stored in the stack area of memory
- Scope
  - available inside the function or block where it is defined
- Life Time
  - only when the function or block where it is defined is executed
- Initialization
  - values are depended by compiler

```
int max(int iA, int iB)
{
    auto int iC ;
    if(iA>iB)
       return iA;
    else
       return iB;
}
```

Extern Variable (外部变量)

- Another name of global variables
- Storage
  - It is stored in the static stored area of memory
- Scope
  - available in the whole program
- Life Time
  - during the execution of the program
- Initialization
  - can be initialized in definition
  - can be initialized by compiler
    - int – 0
    - char – '\0'

Extern Variable (外部变量)

- Extern variable can be available for several C source code files.

File1.c

```
int iA; //definition of external variable
char cB; //definition of external variable
int main()
{
   ...
}
```

File2.c

```
extern int iA; //declaration of external variable
extern char cB; // declaration of external variable
function  (int iX)
{
   ...
}
```

Static Variable(静态变量)

- Static Local Variable (静态局部变量)
  - local variable defined with the keyword static
- Storage
  - It is stored in the static stored area of memory
- Scope
  - available inside the function or block where it is defined
- Life Time
  - during the execution of the program
  - the same with the global variable
- Initialization
  - Only once

```
#include <stdio.h>
void f()
{
 int iA=2;
 static int iB=1;
  iA++;
  iB++;
  printf("iA=%d\n",iA);
  printf("iB=%d\n",iB);
}
int main()
{
  f();
  f();
  return 0;
}
```

```
#include <stdio.h>
void f()
{
  int iA=2;
  static int iB=1;
  iA++;
  iB++;
  printf("iA=%d\n",iA);
  printf("iB=%d\n",iB);
}
int main()
{
  f();
  f();
  return 0;
}
```

Calculate the Factorial Numbers from 1 to 5

```
#include <stdio.h>

int fact(int iN)
{
    static int iFact=1;
    iFact=iFact*iN;
    return (iFact);
}

int main()
{
    int iI;
    for(iI=1;iI<=5;iI++)
      printf("%d!=%d\n",iI, fact(iI));
    return 0;
}
```

1!=1
2!=2
3!=6
4!=24
5!=120

Static Variable(静态变量)

- Static Global Variable (静态全局变量)
  - global variable defined with the keyword static
- Storage
  - It is stored in the static stored area of memory
- Scope
  - available in the source code where it is defined
  - can not be available in other source code
  - different from non-static global variable
- Life Time
  - during the execution of the program
  - the same with the global variable
- Initialization
  - Only once

Registered Variable(寄存器变量)

- Registered variables are stored in register of CPU
- It can be accessed very efficiently
- Since the number of register is limited, only use register variable when it is necessary

```
register iI, iSum=0;
for( iI = 1 ; iI <= 100 ; iI++ )
   iSums=iSum+iI;
printf("iSum=%d\n",iSum);
```

Internal Function & External Function

- Internal Function (内部函数)
  - can only be called inside the source code where it is defined
  - static refers to the limited scope of the function

```
static type function_name (parameter list)
```

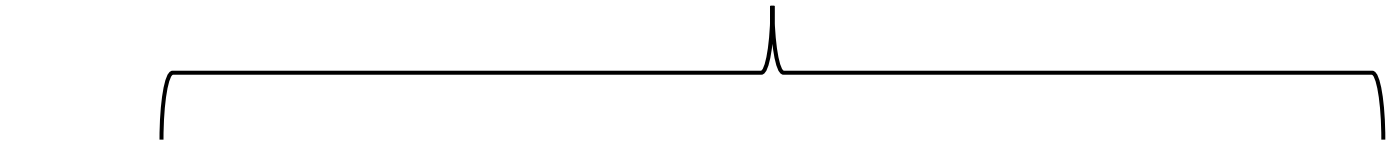
- External Function (外部函数)
  - can be called outside of the source code where it is defined
  - In default, all functions are external functions, so the extern can be omitted

```
extern type function_name (parameter list)
```

Program & Source Codes

- An executable program includes several header files (*.h) and source-code files (*.c).
- Function definitions are written in source-code files, and function declaration are written in header files.
- Only one main function can be defined in one source-code file, the other functions usually defined in the other source code files.
- If the main function and the called function are defined in different source code files, function declaration or including the head file where function declaration exists are required before function calling.

The Executable Program (test.exe in Windows OS)



test.c

```
#include <stdio.h>

//function declaration
int func1();
int func2();

void main()
{
    func1();
    func2();
}
```

funcs.c

```
//definition of func1
int func1()
{
    ......
}

//definition of func1
int func2()
{
    ......
}
```

Since func1 and func2 are defined in another source code file, function declarations are required before the function calling

The Executable Program (test.exe in Windows OS)

test.c

```
#include <stdio.h>
#include “func.h”

void main()
{
    func1();
    func2();
}
```

funcs.c

```
//definition of func1
int func1()
{
    ......
}

//definition of func1
int func2()
{
    ......
}
```

funcs.h

```
//the following macros
are required to avoid
the including of this
head file in multiple
times

#ifndef FUNC_HPP
#define FUNC_HPP

//declaration of func1
int func1();
//declaration of func2
int func2();

#endif
```

Since func1 and func2 are declared in funcs.h, we have to include this header file before these function are called.

- #include <header.h>
  - The header file is provided by the system and stored in the system directory
  - The compiler will firstly search the head file in system directory. If not found, the compiler will search it in the current project directory.

- #include “header.h”
  - The header file is provided by users and stored in the project directory
  - The compiler will firstly search the head file in the current project directory. If not found, the compiler will search in system directory.

- Usually the corresponding header and source-code files have similar filenames
  - XXX.h and XXX.c

Exercise

- Using recursion to write a function to sum all items in an array A = {a_1, a_2, …, a_n}

$$sum(n) = \sum_{i-1}^{n} a_n$$

- The function should be define in a source code file which has no definition of the main function.

test.c

```
#include <stdio.h>
#include "func.h"

#define SIZE 10


int main()
{
    int nums[SIZE];
    int i;
    for ( i = 0 ; i < SIZE ; i++ )
        scanf("%d", &nums[i]);

    printf("sum = %d\n",
         sum(nums ,SIZE));

    return 0;
}
```

func.h

```
#ifndef FUNC_HPP
#define FUNC_HPP

int sum(int nums[], int n);

#endif
```

func.c

```
int sum(int nums[], int n)
{
    if ( n > 1 )
        return nums[n-1]+
                sum(nums, n-1);
    else
        return nums[0];
}
```

Sequential Searching(顺序查找)

- The simplest searching method

- Search the required element one by one in a list (ex. comparing each element in an array)
  - Successful : return the index of the element
  - Failed : visit all element

```
int search (int a[], int n, int key)
{
    int i;
    int index = -1;
    for ( i = 0 ; i < n ; i++ )
        if ( key == a[i] )
            index = i;
    return index;
}
```

Example of Sequential Searching

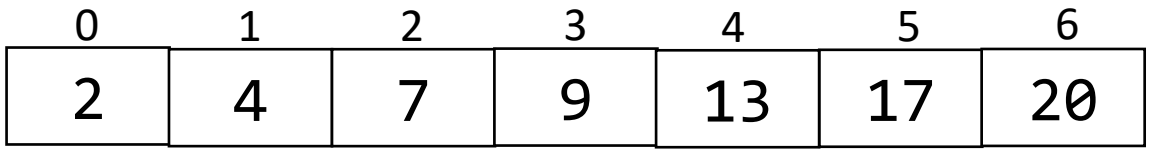
```
#include <stdio.h>
#define N 6
int search (int a[], int n, int key)
{
    int i;
    int index = -1;
    for ( i = 0 ; i < n ; i++ )
        if ( key == a[i] )
            index = i;
    return index;
}

int main()
{
    int a[N] = {6, 3, 8, 4, 7, 5};
    int ind, key = 4;
    ind = search(a, N, key);
    if (res != -1) printf(“found! index is %d\n”, ind);
    else printf(“not found!\n”);
    return 1;
}
```

Binary Searching(二分法查找)

- Pre-request
  - data must be stored in an ordered list(有序列表)

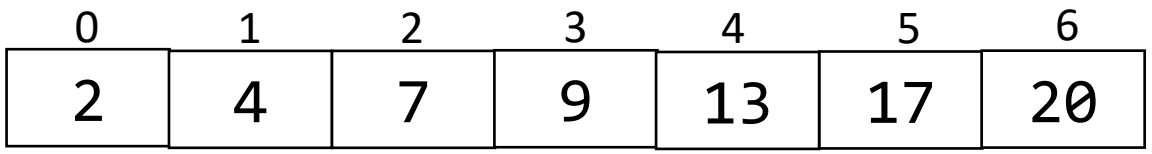
```
int a[7] = {2, 4, 7, 9, 13, 17, 20};
int low, high, middle;
int key = 7;
```

ROUND-1



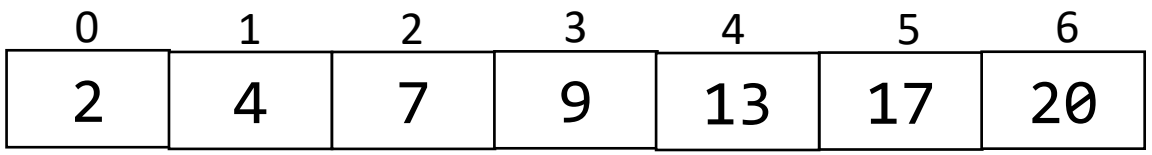
compare key and a[middle] : key < a[3]

ROUND-2



compare key and a[middle] : key > a[1]

ROUND-3



high = low = middle

compare key and a[middle] : key == a[2] (found!)

Binary Searching(二分法查找)

- Given a list with ascending order(a[N])
  - Compare the key with the middle element.
  - If equal, find the key
  - If not, update the low or high index according to the comparison
    - if key < a[middle] : high = middle – 1
    - if key > a[middle] : low = middle + 1

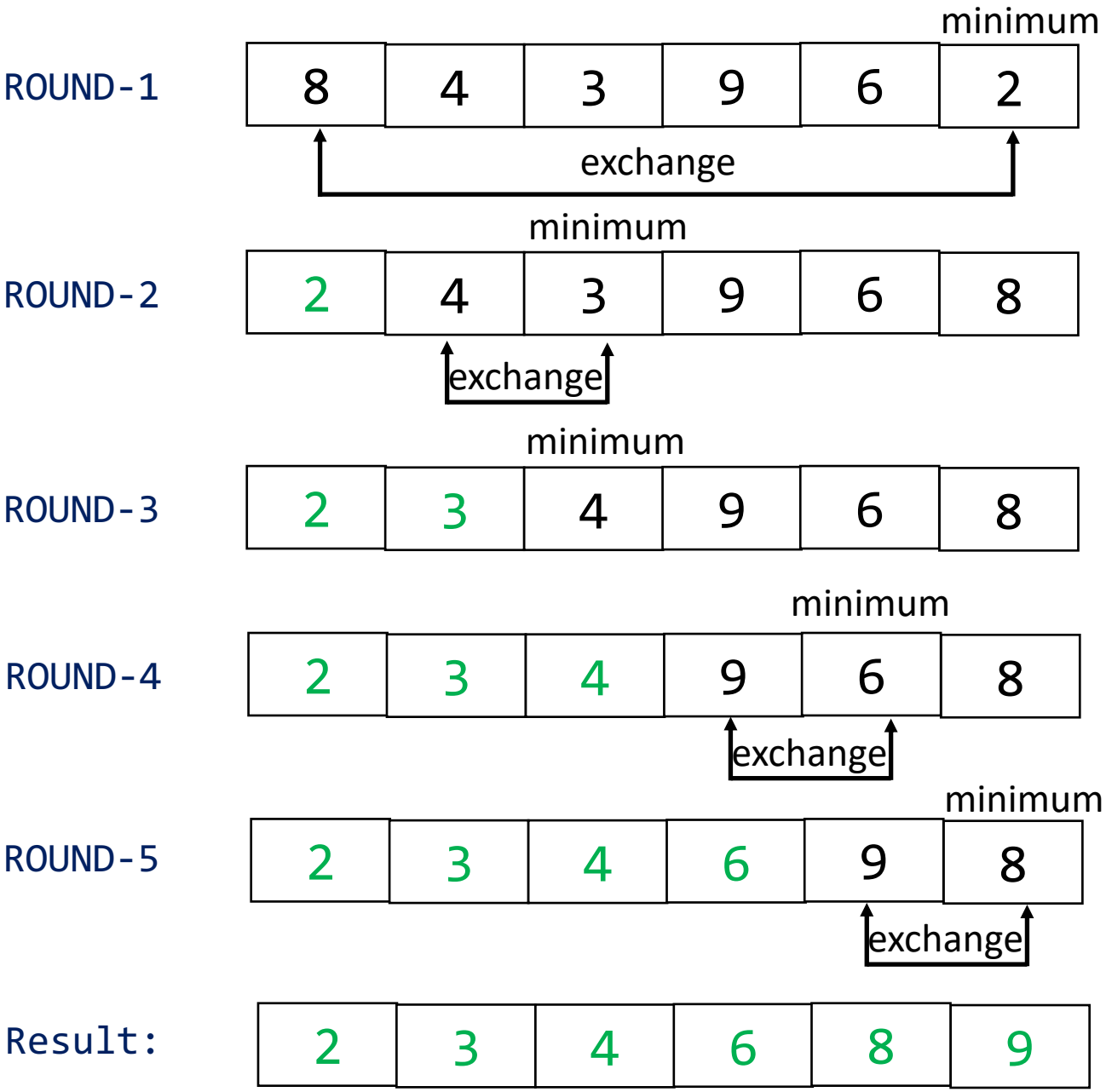
```
Input :  a[N] (ordered array)
         key
Output: index

low = 0, high = N-1 //initialization
index = -1 // not found

while(low<=high):
    middle = (low + hight) / 2
    if (key == a[middle]):
        index=middle
        break the loop
    else:
        if (a[middle] > key):
            high = middle – 1
        else:
            low = middle + 1;
```

```
int binarySearch(int a[], int n, int key)
{
    int middle, low = 0, high = n-1;
    int index = -1;
    while ( low <= high )
    {
        middle = (low + high) / 2;
        if ( a[middle] == key )
        {
            index = middle;
            break;
        }
        else
        {
            if ( a[middle] > key )
                high = middle – 1;
            else
                low = middle + 1;
        }
    }

    return index;
}
```

Selective Sorting (选择排序)

- Find the minimal element in a list (N elements), and exchange it with the first element
- Find the minimal element in the rest N-1 elements, and exchange it with the 2nd element
- Find the minimal element in the rest N-2 elements, and exchange it with the 3rd element.
- ……

```
int a[6] = {8, 4, 3, 9, 6, 2};
```



```
void selectSort(int a[], int n)
{
    int i, j;
    int minIndex;
    int temp;

    for ( i = 0 ; i < n-1 ; i++ )
    {
        minIndex = i;
        for ( j = i+1 ; j < n ; j++ )
            if ( a[j] < a[minFlag] )
                minIndex = j;

        if ( i != minIndex )
        {
            temp = a[minIndex];
            a[minIndex] = a[i];
            a[i] = temp;
        }
    }
}
```