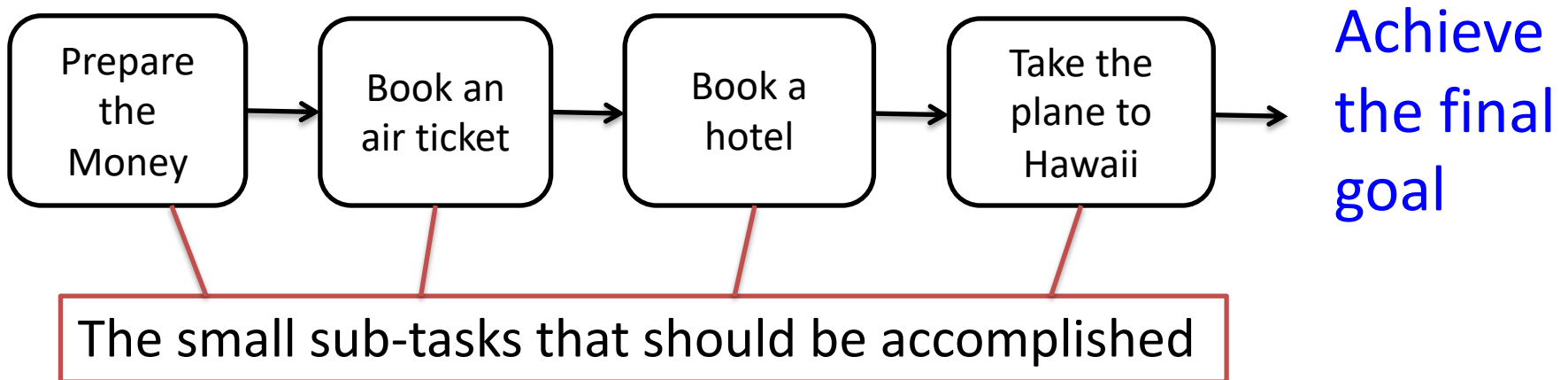


Class 7 Function(函数)-1

Introduction

- How to realize a goal?
 - Analyze the goal and divide it into a set of smaller and easier sub-tasks.
- Example : How can I make a trip to Hawaii?



Introduction

- Modulus programming
 - Organize a large program into small, independent program segments that are **Modules**
 - Modules separately named and individually callable program units.
- **Function** in C Language
 - Each module can be implemented as a function that is responsible for a single task
 - By using functions, modules can be reused and programming codes are easier to be maintained

Function and C Program

- C program is comprised by different functions each of which accomplish a small task
- Three Types of Functions
 - User-defined Function : ex. gcd(), lcm()
 - Library Function : ex. printf(), scanf()
 - Main Function
- Main function is the entry of each program, and the program is finished when the main function is finished.
- Each program can include several C code files, but it only has **one main function** that are written in one C code file.

Three Important Concepts of Function

- Function Definition (函数定义)
- Function Calling (函数调用)
- Function Declaration (函数声明)

Function Definition

Function Header	<code>type function_name(parameter list)</code>
Function Body	<pre>{ declaration of local variables; statements; }</pre>

Function Header(函数头)

`type function_name(parameter list)`

- Function type (returning type)
 - Type of return value
 - Assume to be **int** if not specified
 - **void** : return nothing
 - Code **explicitly** the return type, even when it is an integer.
- function name
 - Legal Identifier
- Formal parameter list
 - No value at the time declaration
 - Obtained values when being called.
 - Can be omitted if there are no parameters

Examples for Headers

```
float quadratic(int a,int b,int c){...}
```

```
double power(double x, int n){...}
```

```
float mul(float x,float y){...}
```

```
int sum(int a,int b){...}
```

```
void printline(void){...}
```

```
void printline(){...}
```


Function Body (函数体)

- Local variables declarations
- Function statements
- Return statement (Can be omitted if the function returns nothing.)

```
float mul (float x, float y)
{
    float result; /*local variable*/
    /*compute the product*/
    result = x * y;
    /*return the result*/
    return result;
}
```

Notes for Function Definition

- A local variable is the variable that can **only** be used inside a function
- When a function reaches its return statement, the control is **immediately** transferred back to the calling program.
- In the absence of a return statement, the closing brace acts as a void return.

```
#include <stdio.h>
#include <time.h>
```

```
long showTime()
```

```
{
    long lct;
    lct = time(0);
    return lct;
}
```

```
int main()
{
    long lCurTime;
    lCurTime = showTime();
    printf("%ld\n", lCurTime);
    return 0;
}
```

Function Header:

No parameter list

The function return a long-type value

Function Body:

The part between the brace { }

lct is a local variable and only can be used inside the function

Call the Function:

The program will jump into the body of the showTime() function, and the printf() will be executed when the showTime() function is finished.

No Returned Value

- If a function has no returned value, the keyword of void should be used in the function header

```
#include <stdio.h>
#include <time.h>

void showTime()
{
    long lct;
    lct = time(0);
    printf("%ld\n", lct);
    //return;
}

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

Call a Function

`function_name(parameter list)`

- Examples

```
float a = 3.0f, b = 5.0f, c;  
.....  
//call a function without parameter list  
showTime();  
  
//call a function with parameter list  
mul(3.0f, 5.0f);  
mul(a, 5.0f);  
mul(a, b);  
  
//call a function and assign the returned value  
//to a variable  
c = mul(a, 9.0);
```

Jump into a Function

```
ln-01  #include <stdio.h>
ln-02  #include <time.h>
ln-03
ln-04  long showTime()
ln-05  {
ln-06      long lct;
ln-07      lct = time(0);
ln-08      return lct;
ln-09  }
ln-10
ln-11  int main()
ln-12  {
ln-13      long lCurTime;
ln-14      lCurTime = showTime();
ln-15      printf("%ld\n", lCurTime);
ln-16      return 0;
ln-17  }
```



Function Prototype (函数原型)

- Two Forms of Function Prototype

```
type function_name(type param_name1, type param_name2, ...);
```

OR

```
type function_name(type, type, ...);
```

- Function prototype is required in the function declaration(函数声明)

Function Declaration

- If a function is defined after the position where it is called, it should be declared before it is called.
- Function declaration inform the compiler the function prototype that is called.
- Examples of Function Declaration

```
// all parameter names are given in the declaration
int mul(int m, int n);
// the parameter names can be omitted
int mul(int, int);
// another example
long showTime();
```


Function Declaration is
not required

```
#include <stdio.h>

int add(int a, int b)
{
    return a+b;
}

int main ()
{
    int a=1, b=2, c;
    c = add(a,b);
    printf("%d\n", c);
    return 0;
}
```

Function Declaration should
be used

```
#include <stdio.h>
//Function Declaration
int add(int a, int b);

int main ()
{
    int a=1, b=2, c;
    c = add(a,b);
    printf("%d\n", c);
    return 0;
}

int add(int a, int b)
{
    return a+b;
}
```

Other Examples of Function Declaration

```
#include <stdio.h>
//Function Declaration
int add(int, int);

int main ()
{
    int a=1, b=2, c;
    c = add(a,b);
    printf("%d\n", c);
    return 0;
}

int add(int a, int b)
{
    return a+b;
}
```

```
#include <stdio.h>

int main ()
{
    int a=1, b=2, c;
    //Function Declaration
    int add(int, int);
    c = add(a,b);
    printf("%d\n", c);
    return 0;
}

int add(int a, int b)
{
    return a+b;
}
```

Notes for function declarations

- The parameter list must be separated by **commas**.
- The parameter **names do not need** to be the same in the prototype declaration and the function definition.
- Using parameter names in the declaration is **optional**.
- The **types must match** the types of parameters in the function definition, in number and order. Otherwise, a compiler may produce an error.
- The returned type is optional when the function returns **int** type data.
- The returned type **must be void** if no value is returned.

Notes for function declarations

- A prototype declaration may be placed in two places in a program:
 - Above all the functions (including *main*)
 - as a *global prototype*: the function is available to all in this file.
 - Inside a function definition.
 - as a *local prototype*: the function is only visible where the declaration locates.

Any Problem of the Following Function Declaration?

```
int fun1(int, int);
```

```
int main ()  
{  
    int a, b, c;  
    void fun2(int, int);  
    fun2(a,b);  
    c = fun1(a, b);  
}
```

fun2 is not available in
the definition of fun1()

```
int fun1(int m, int n) {...fun2(m,n);...}  
void fun2(int m, int n) {...}
```

Function Declaration by a global way is correct for the following program

```
int fun1(int, int);  
void fun2(int, int);  
  
int main ()  
{  
    int a, b, c;  
    fun2(a,b);  
    c = fun1(a, b);  
}  
  
int fun1(int m, int n) {...fun2(m,n);...}  
void fun2(int m, int n) {...}
```

Exercise

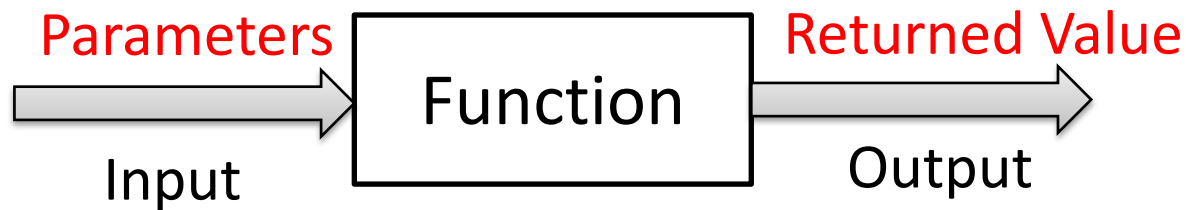
- Define a function to calculate the area of a circle and calculate the area of a circle whose radius is 5.0

```
#include <stdio.h>
#define PI 3.1415926

float calArea(float radius)
{
    return PI*radius*radius;
}

int main()
{
    float radius = 5.0f;
    float area = calArea(radius);
    printf("Area = %f\n", area);
    return 0;
}
```

Parameters and Returned Value



- Parameters
 - Through **formal parameters** (形式参数) and **actual parameters** (实际参数), values are passed into a function for calculation
- Returned Value
 - Through the returned value, the calculated result (value) are outputted.


```
#include <stdio.h>

double max(double, double);

int main()
{
    double a = 3.0, b = 5.0, c;
    c = max( a, b );
    printf("Max is %f\n", c);
    printf("Max is %f\n", max( c, 3.5 ) );
    return 0;
}

double max(double x, double y)
{
    int z;
    z = x > y ? x : y;
    return(z);
}
```

The diagram illustrates the execution flow of the provided C code. It features two functions: `main` and `max`. In the `main` function, there are two calls to the `max` function. The first call is `c = max(a, b);`, where `a` and `b` are `double` variables. The second call is `printf("Max is %f\n", max(c, 3.5));`, where `c` is a `double` variable and `3.5` is a `double` literal. The `max` function is defined as `double max(double x, double y)`, where `x` and `y` are `double` parameters. The diagram uses orange arrows to show the flow of arguments from the function calls to the parameters of the `max` function. A red arrow shows the return value of the `max` function being passed back to the `main` function. A red oval highlights the `main` function's scope.

Formal and Actual Parameters

- Formal Parameters (形式参数、形参)
 - Parameters used in the **function definition**

```
double max(double x, double y) {...}
```

- Actual Parameters (实际参数、实参)
 - Parameters used in **function calling**
 - can be expression, constant or variables

```
double a = 3.0, b = 5.0;  
max(a, b);  
max(a, 7.1);  
max(a+1.5, b);
```

Notes of Formal and Actual Parameters

- The values of **actual parameters** are passed to corresponding **formal parameters** when the function are called
- Actual parameters should be compatible to formal parameters
- Memory of formal parameters is allocated only when the function is called
- The values can ONLY be passed **from actual parameters to formal parameters**. The inverse passing is IMPOSSIBLE

Memory in a C Program

- Static Stored Area (静态存储区)
 - The memory in the static stored area is allocated in the compilation of the source code. This memory is held during the execution of the program. Static data, global variable and constant data are stored in this part
- Stack Area (栈区)
 - The memory in the stack area is allocated when a function is called, and released when the function is finished. Variables of functions are stored in this part.
- Heap Area(堆区)
 - The memory in the heap area is allocated by users. It is allocated (分配) by `mollac()`, and released(释放) by `free()`

```

#include <stdio.h>

double max(double, double);

int main()
{
    double a = 3.0, b = 5.0, c;
    c = max( a, b );
    printf("Max is %f\n", c);
    return 0;
}

double max(double x, double y)
{
    int z;
    z = x > y ? x : y;
    return z;
}

```

Program Memory

	stack area (栈区)	
0xAB00	3.0	a
0xAB08	5.0	b
0xAB10		c
	.	
	.	
	.	
0xAB70		
0xAB78		
0xAB80		
0xABCD		

```

#include <stdio.h>

double max(double, double);

int main()
{
    double a = 3.0, b = 5.0, c;
    c = max( a, b );
    printf("Max is %f\n", c);
    return 0;
}

double max(double x, double y)
{
    int z;
    z = x > y ? x : y;
    return z;
}

```

Program Memory

	stack area (栈区)	
0xAB00	3.0	a
0xAB08	5.0	b
0xAB10		c
	.	
	.	
	.	
0xAB70	3.0	x = a
0xAB78	5.0	y = b
0xAB80	5.0	z
0xABCD		

Formal parameters are allocated when the function is called

Values are passed from the formal parameters to the actual parameters

```


#include <stdio.h>

double max(double, double);

int main()
{
    double a = 3.0, b = 5.0, c;
    c = max( a, b );
    printf("Max is %f\n", c);
    return 0;
}

double max(double x, double y)
{
    int z;
    z = x > y ? x : y;
    return z;
}

```



Program Memory

	stack area (栈区)		Memory of returned value is allocated when the value is returned.
0xAB00	3.0	a	
0xAB08	5.0	b	
0xAB10	5.0	c	
	.		Memory allocated in stack area will be released when the function is finished
	.		
	.		
0xAB70	3.0	x	
0xAB78	5.0	y	
0xAB80	5.0	z	
0xABCD	5.0	returned value	
	.		
	.		

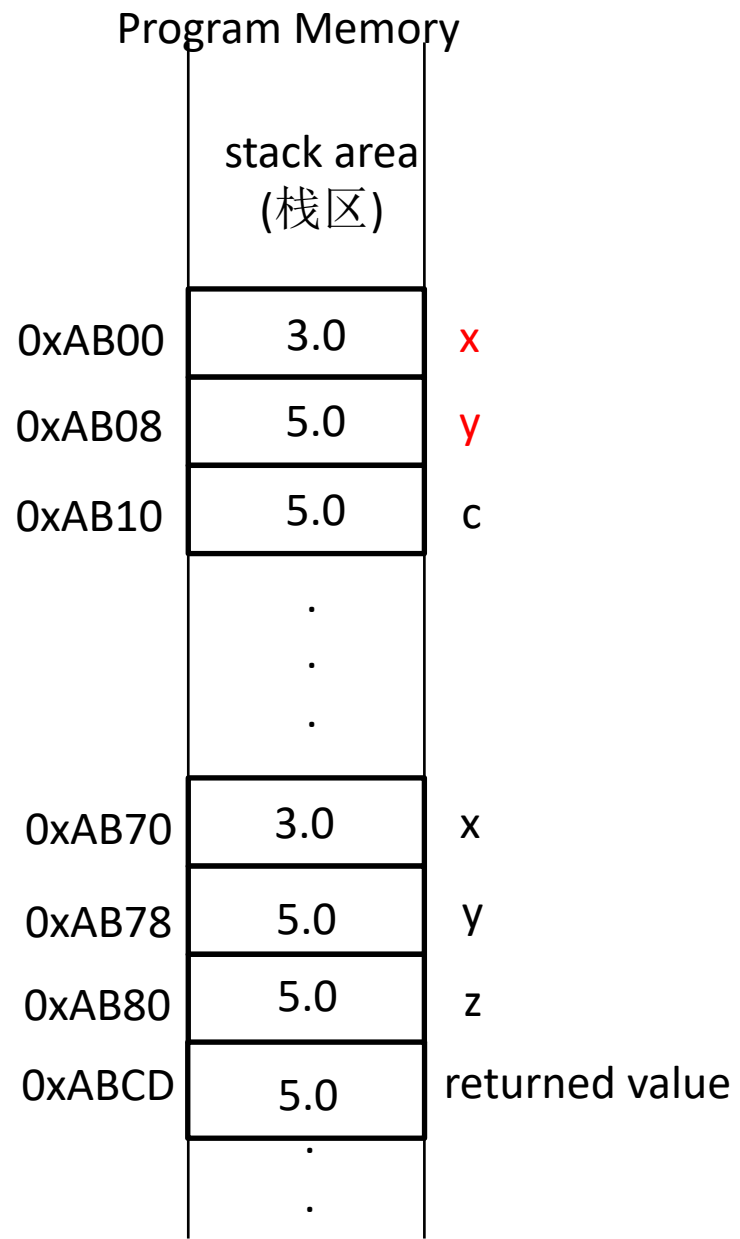
Exercise : How about the memory allocation in this program?

```
#include <stdio.h>

double max(double, double);

int main()
{
    double x = 3.0, y = 5.0, c;
    c = max( a, b );
    printf("Max is %f\n", c);
    return 0;
}

double max(double x, double y)
{
    int z;
    z = x > y ? x : y;
    return z;
}
```




Are the values of x and y swapped?

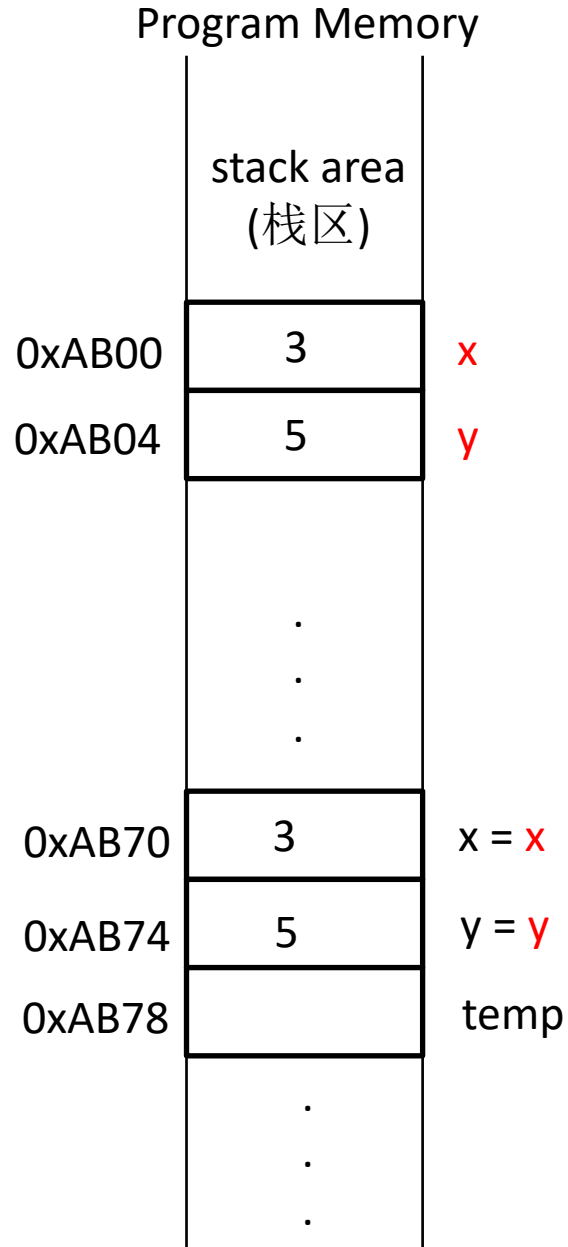
```
#include <stdio.h>

int swap(int, int);
int main()
{
    int x = 3, y = 5;
    swap( x, y );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```



Output: $x = 3, y = 5$




Are the values of x and y swapped?

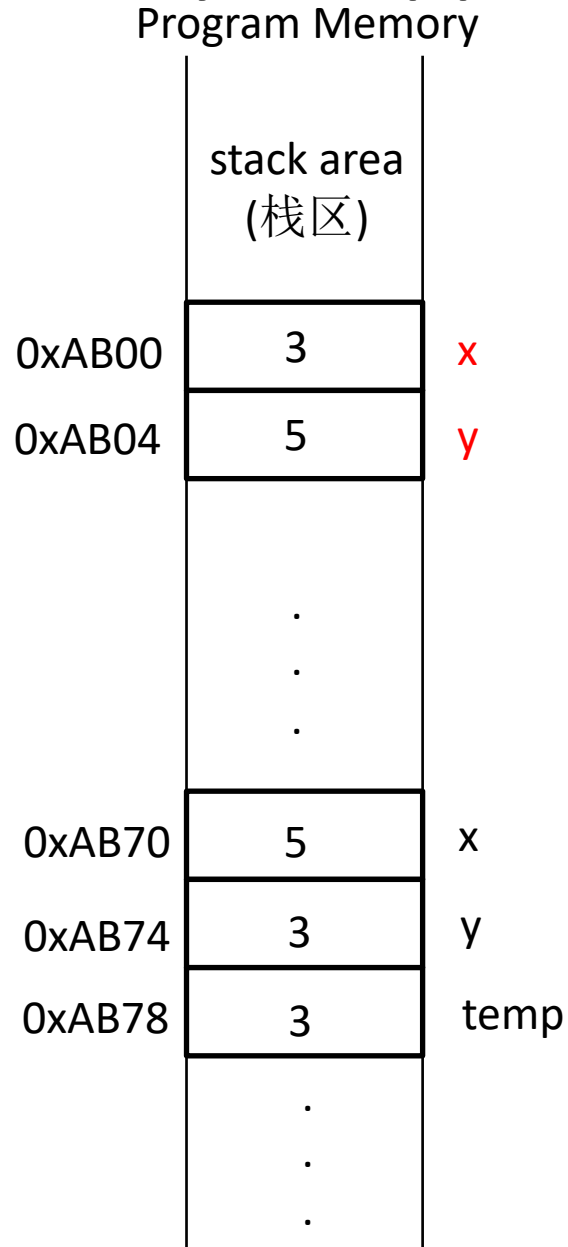
```
#include <stdio.h>

int swap(int, int);
int main()
{
    int x = 3, y = 5;
    swap( x, y );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```



Output: $x = 3, y = 5$



Are the values of x and y swapped?

```
#include <stdio.h>

int swap(int, int);
int main()
{
    int x = 3, y = 5;
    swap( x, y );
    printf("x=%d, y=%d\n",x,y);
    return 0;
}

void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Output: $x = 3, y = 5$

Program Memory

stack area (栈区)	
0xAB00	3
0xAB04	5
	.
	.
	.
0xAB70	
0xAB74	
0xAB78	
	.
	.
	.

Values of Actual
Parameters can
not be Changed
by using
a Function

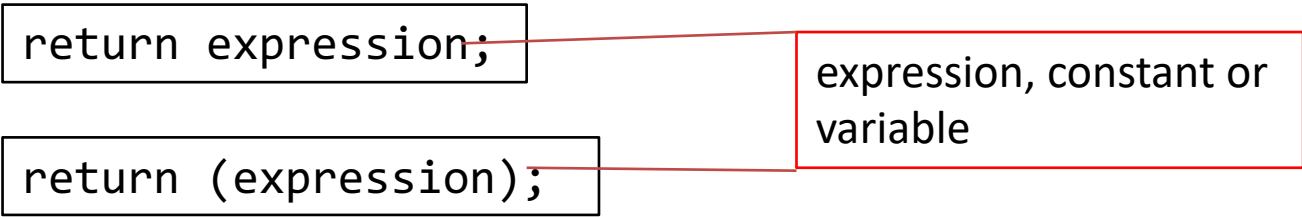
Returned Value

OR

```
return expression;
```

```
return (expression);
```

expression, constant or
variable

A diagram illustrating two equivalent ways to write a return statement. Two boxes, one containing 'return expression;' and the other 'return (expression);', are connected by red lines to a single box on the right containing 'expression, constant or variable'.

- There could be several return statement in a function, but only one return statement can be operated
- When the return statement is operated, the function is finished and the execution will return to the caller.

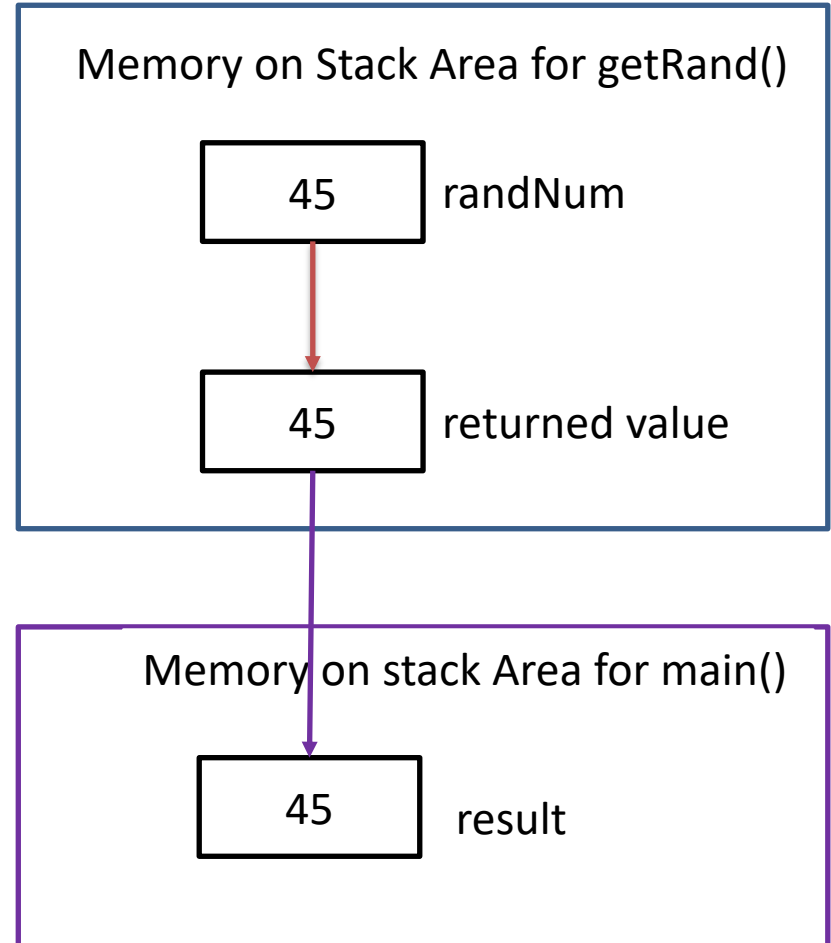
Example : calculate a random number and output it

```
#include <stdio.h>
#include <time.h>

int getRand();

int main()
{
    int result;
    result = getRand();
    printf("%d\n", result);
    return 0;
}

int getRand()
{
    int randNum;
    srand(time(0));
    randNum = rand()%100+1;
    return randNum;
}
```



Returned Value

- A function can only return ONE value
- If no type is given in function definition, a function will return a value of **int** type.
- If a function return no value, **void** can be used in function definition
- The returned value should be the same type as what is defined in the function. If different, the returned value will automatically casted to the type mentioned in function definition

Type casting also automatically be performed when a function is called

Example : Type Casting

```
#include <stdio.h>

float add1 ( float x , float y )
{
    return ( x + y ) ;
}

double add2 ( int i , int j )
{
    return ( i + j ) ;
}

int main ( )
{
    double a =3 , b = 3.2, sum;
    sum = add1 ( a , b );
    sum = add2 ( a , b );
    return 0;
}
```

The result of (i + j) is a value of **int** type. It will be casted to a **double** type and returned

a and b are casted to **float** type

a and b are casted to **int** type

Array as Function Parameters

- Array Elements as Function Parameters (Passing Value)
 - This is the same as using variables as function parameters
 - Values of array elements are passed into the function

```
#include <stdio.h>

void fun(int x, int y, int z)
{
    x++; y++; z++;
    printf("%d %d %d\n", x, y, z);
}

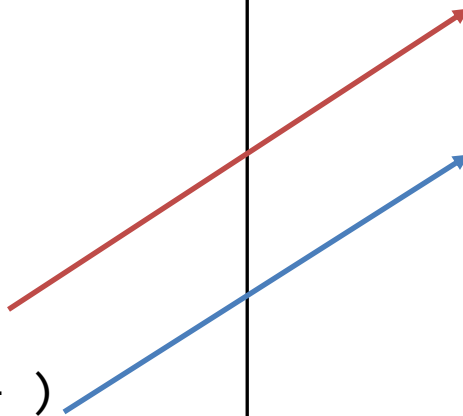
int main ( )
{
    int a[3] = {1, 2, 3}, i;
    fun( a[0], a[1], a[2] );

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);
    printf("\n");
    return 1;
}
```

Output:

2 3 4

1 2 3




```

#include <stdio.h>

void fun(int x, int y, int z)
{
    x++; y++; z++;
    printf("%d %d %d\n", x, y, z);
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;
    fun( a[0], a[1], a[2] );
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);
    printf("\n");
    return 1;
}

```

Program Memory

stack area (栈区)		
0xAB00	1	a[0]
0xAB04	2	a[1]
0xAB08	3	a[2]
	.	
	.	
	.	
0xAB70		
0xAB74		
0xAB78		
	.	
	.	
	.	



```
#include <stdio.h>

void fun(int x, int y, int z)
{
    x++; y++; z++;
    printf("%d %d %d\n", x, y, z);
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a[0], a[1], a[2] );

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}
```

Program Memory

stack area (栈区)		
0xAB00	1	a[0]
0xAB04	2	a[1]
0xAB08	3	a[2]
	.	
	.	
	.	
0xAB70	1	x = a[0]
0xAB74	2	y = a[1]
0xAB78	3	z = a[2]
	.	
	.	
	.	



```
#include <stdio.h>

void fun(int x, int y, int z)
{
    x++; y++; z++;
    printf("%d %d %d\n", x, y, z);
}

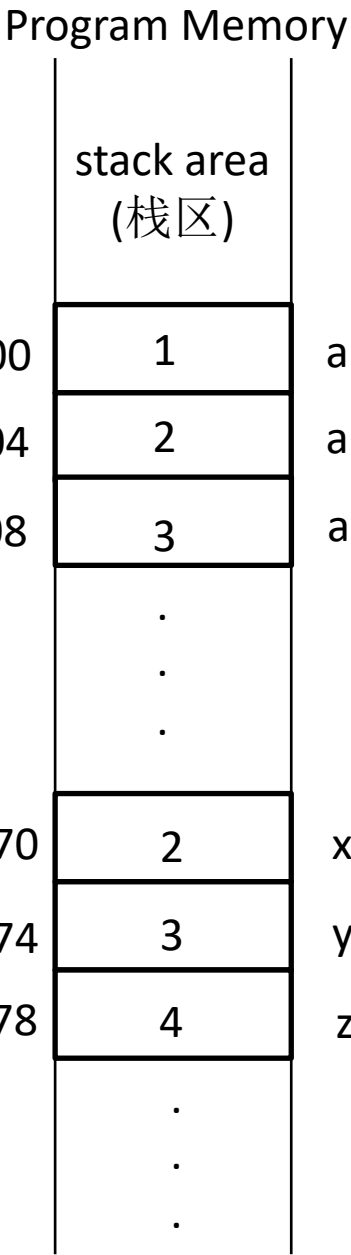
int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a[0], a[1], a[2] );

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}
```



```

#include <stdio.h>

void fun(int x, int y, int z)
{
    x++; y++; z++;
    printf("%d %d %d\n", x, y, z);
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a[0], a[1], a[2] );

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory

stack area (栈区)		
0xAB00	1	a[0]
0xAB04	2	a[1]
0xAB08	3	a[2]
	.	
	.	
	.	
0xAB70		
0xAB74		
0xAB78		
	.	
	.	
	.	

Array as Function Parameters

- Array Names as the Function Parameters (Passing Address)
 - The array name is constant address of that array
 - When the array name is used as actual parameter, only the address is passed into the function
 - The memory on the stack area is allocated to store the address, not the whole array. By using this address, the value of each array element can be changed

```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;
    fun( a );

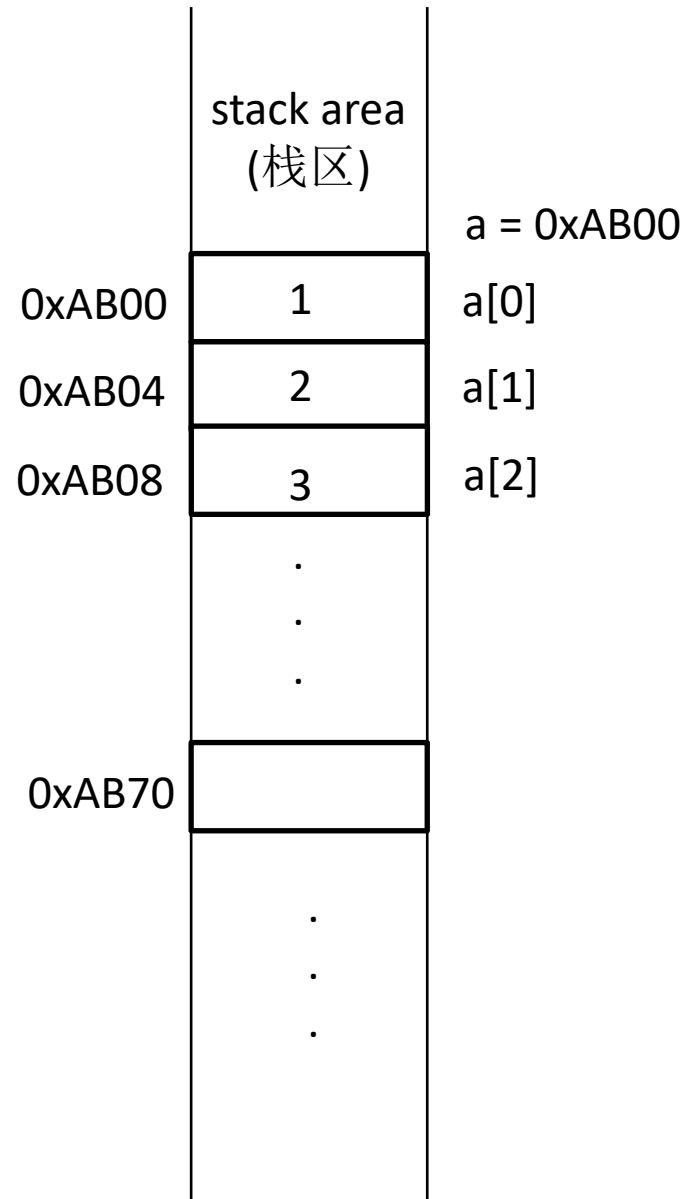
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory



```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a );

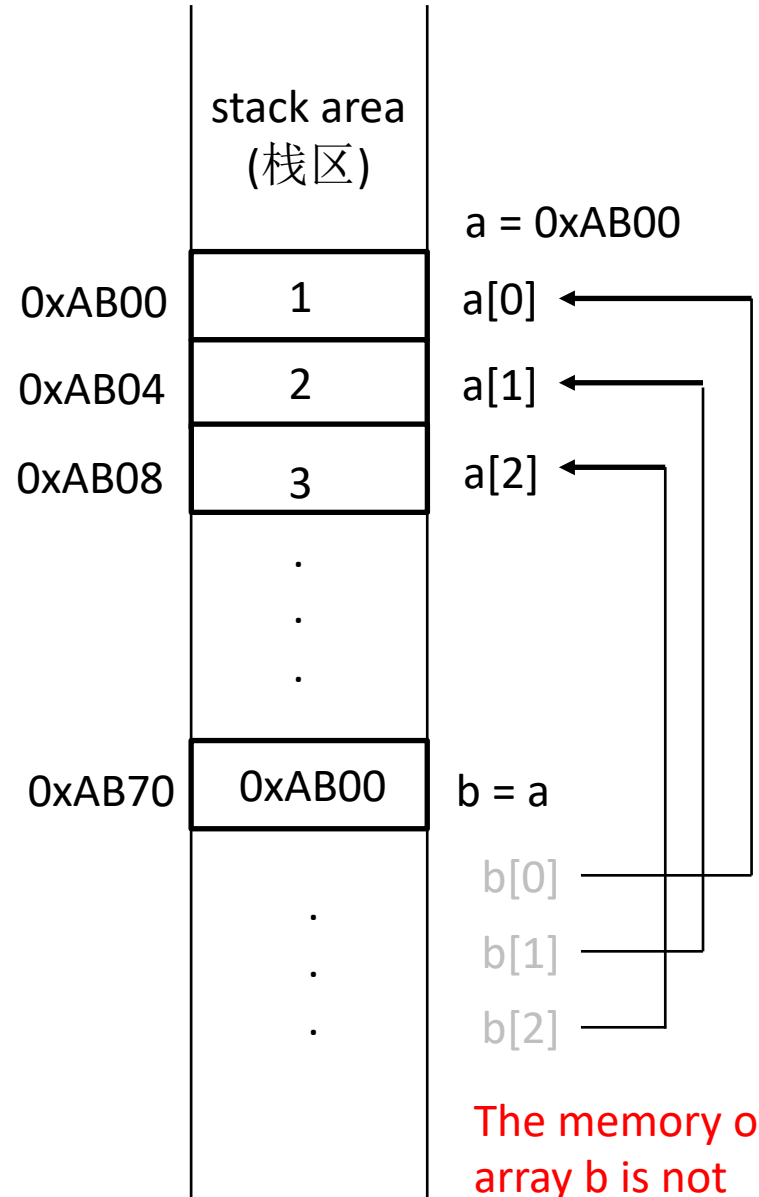
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory



The memory of array b is not allocated on stack

```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a );

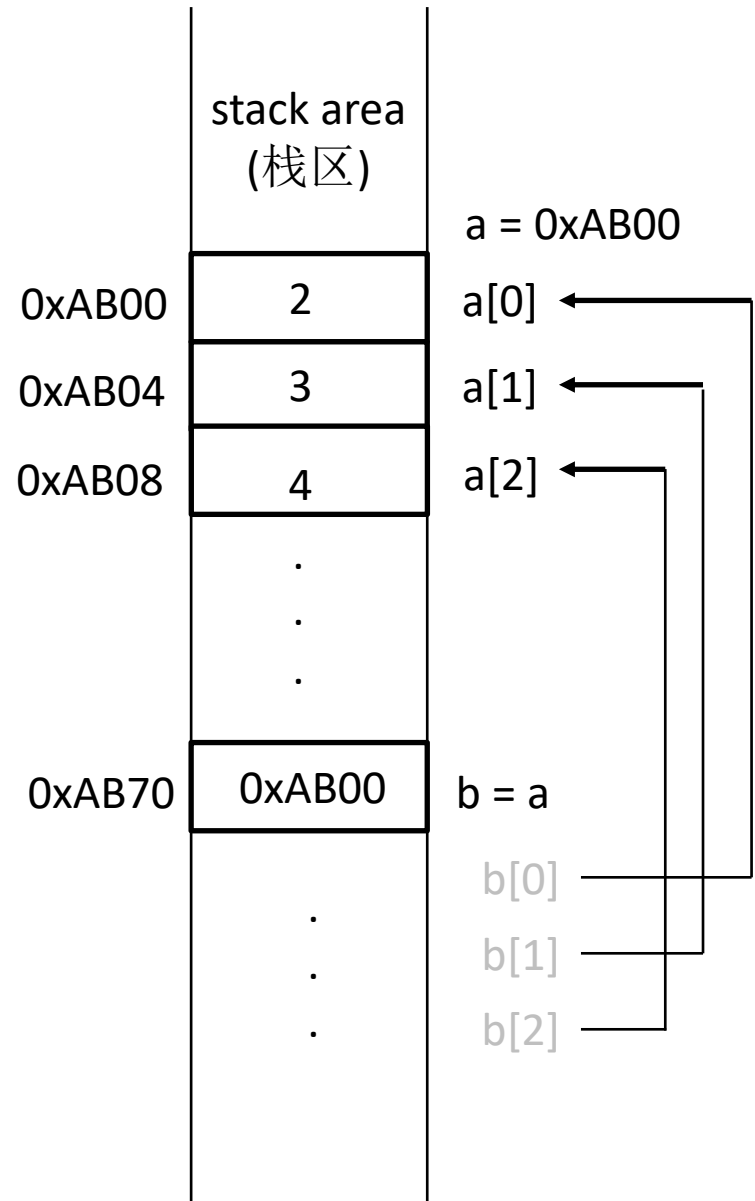
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory




```

#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a );

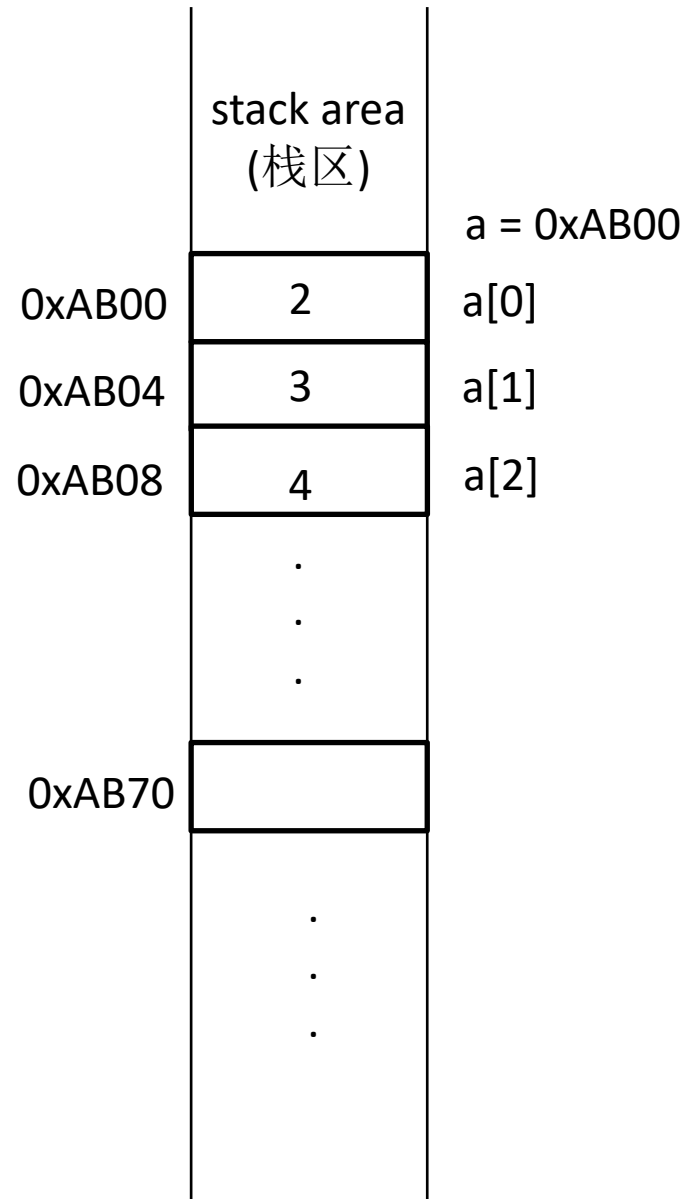
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);

    printf("\n");

    return 1;
}

```

Program Memory



```
#include <stdio.h>

void fun(int b[])
{
    int i;
    for ( i = 0 ; i < 3 ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;
    fun( a );
    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);
    printf("\n");
    return 1;
}
```

Output:

2 3 4

2 3 4

- When array name is used as the function parameter, usually another int type variable is used to indicate the length of the array

```
#include <stdio.h>

void fun(int b[], int length)
{
    int i;
    for ( i = 0 ; i < length ; i++ )
        printf("%d ", ++b[i]);
    printf("\n");
}

int main ( )
{
    int a[3] = {1, 2, 3}, i;

    fun( a, 3);

    for( i = 0 ; i < 3 ; i++ )
        printf("%d ", a[i]);
    printf("\n");

    return 1;
}
```

Exercise

- Design a function to calculate the Fibonacci sequence, and design another function to output all elements of the sequence
 - Use array as the function parameter

$$f(n) = \begin{cases} 1, & n = 0 \\ 1, & n = 1 \\ f(n-1) + f(n-2), & \text{others} \end{cases}$$

```
void Fib(int a[], int n);  
void print(int a[], int n);
```

```

#include <stdio.h>
#define N 10

void Fib(int a[], int n)
{
    int i;
    if (n < 1) return;
    else if (n == 1) {a[0] = 1; return;}
    else if (n == 2) {a[1] = 1; return;}
    else
    {
        a[0] = 1; a[1] = 1;
        for ( i = 2 ; i < n ; i++ )
            a[i] = a[i-2] + a[i-1];
        return;
    }
}

.....

```

```

.....
void print(int a[], int n)
{
    int i;
    for ( i=0 ; i<n ; i++ )
        printf("%d ", a[i]);
    printf("\n");
}

int main()
{
    int a[N];
    Fib(a, N);
    print(a, N);
    return 0;
}

```