

C/C++ Program Design

LAB 6

CONTENTS

- Master how to use the Library Function
- Master how to declare, define, and call a User-Defined Function

2 Knowledge Points

2.1 Library Function

2.2 User-Defined Function

2.3 Recursive function

2.4 Pointers to functions

2.1 Library Function

Example: Library Function

```
#include <iostream>
#include <cmath> // header file
using namespace std;

int main() {
    double number, squareRoot;

    cout << "Enter a number:";
    cin >> number;

    // sqrt() is a library function to calculate square root
    squareRoot = sqrt(number); // sqrt() is a library function
    cout << "Square root of " << number << " = " << squareRoot;

    return 0;
}
```

Output:

Enter a number:25

Square root of 25 = 5

2.2 User-Defined Function

Syntax of defining a function:

function header

```
return_type function_name (datatype parameter1, datatype parameter2, ...)  
{  
    // function body  
}
```

- **return type:** suggests what type the function will return. It can be **int, char, string, pointer** or even a class **object**. If a function does not return anything, it is mentioned with **void**.
- **function name:** is the name of the function, using the function name it is called.
- **parameters:** are variables to hold values of arguments passed while function is called. A function may or may not contain parameter list(**void**).

Function prototype:

The simplest way to get a prototype is to copy the **function header** and add a **semicolon**.

Here are some function prototypes:

```
// A function takes two integers as its parameters  
// and returns an integer  
int max(int, int);
```

```
// A function takes a char and an integer as its parameters  
// and returns an integer  
int fun(char, int);
```

```
// A function takes a char as its parameter  
// and returns a pointer-to-char  
char *call(char );
```

```
// A function takes a pointer-to-int and an integer  
// as its parameters and returns a pointer-to-int  
int *swap(int *, int);
```

Example: Declaring, Defining and Calling a function

```
git sumfunction.cpp > ...  
1 #include <iostream>  
2 using namespace std;  
3  
4 //Declaring a function  
5 int sum(int x, int y);  
6 int main()  
7 {  
8     int a = 10;  
9     int b = 20;  
10    int c;  
11  
12    //Calling a function  
13    c = sum(a,b);  
14  
15    cout << a << " + " << b << " = " << c << endl;  
16  
17    return 0;  
18 }  
19  
20 // Defining a function  
21 int sum(int x, int y)  
22 {  
23     return (x + y);  
24 }
```

Declaring a function (function prototype)

Calling a function

Defining a function
Outside from all functions

Output:

```
10 + 20 = 30
```

Actual parameter and Formal parameter

sumfunction.cpp > ...

```
1 #include <iostream>
2 using namespace std;
3
4 //Declaring a function
5 int sum(int x, int y);
6 int main()
7 {
8     int a = 10;
9     int b = 20;
10    int c;
11
12    //Calling a function
13    c = sum(a,b);
14
15    cout << a << " + " << b << " = " << c << endl;
16
17    return 0;
18 }
19
20 // Defining a function
21 int sum(int x, int y)
22 {
23     return (x + y);
24 }
```

Actual parameters(arguments)

When calling a function, the values of arguments are assigned to the parameters

Formal parameters

```
↳ sumfunction.cpp > ...
```

```
1 #include <iostream>
2 using namespace std;
3
4 //Declaring a function
5 int sum(int x, int y);
6 int main()
7 {
8     int a = 10;
9     int b = 20;
10    int c;
11
12    //Calling a function
13    c = sum(a,b);
14
15    cout << a << " + " << b << " = " << c << endl;
16
17    return 0;
18}
19
20 // Defining a function
21 int sum(int x, int y)
22 {
23     return (x + y);
24}
```

```
30
```

Note:

- The return type is **int**, so, **c** is of type **int**.
- The return type of function is defined in function declaration in `sum(int x, int y);`
- If no value is returned, **void** should be used.

作用域

生存区

Scope and duration of variable

An variable's **scope** is where the variable can be referenced in a program. Some identifiers can be referenced throughout a program, others from only portions of a program.

A variable defined inside a function is referred to as a **local variable**.
A **global variable** is defined outside functions.

An variable's **storage duration** is the period during which that variable exists in memory.

```

int a;
void main()
{
    .....
    .....
    f2;
    .....
    f1;
    .....
}
f1()
{
    auto int b;
    .....
    f2;
    .....
}
f2()
{
    static int c;
    .....
}

```



scope of a

duration of a:

duration of b:

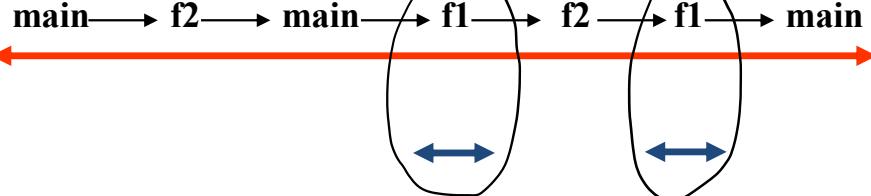
duration of c:

scope of b

scope of c

~~局部变量都是 auto~~

~~全局变量的生存期，局部变量的作用域。~~



具有全局变量的生存期，局部变量的作用域。

1. Passing arguments to a function **by value**

```
passvalue.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 void swap(int x, int y)
5 {
6     int z;
7     z = x;
8     x = y;
9     y = z;
10 }
11
12 int main()
13 {
14     int a = 45, b = 35;
15     cout << "Before Swap\n";
16     cout << "a = " << a << ",b = " << b << endl;
17
18     swap(a,b);
19
20     cout << "After Swap\n";
21     cout << "a = " << a << ",b = " << b << endl;
22
23     return 0;
24 }
```

before calling:

a: 45

b: 35

calling: *A COPY*

a: 45

b: 35

x: 45

y: 35

x: 35

y: 45

z:

after calling:

a: 45

b: 35

Output:

Before Swap

a = 45, b = 35

After Swap

a = 45, b = 35

2. Passing arguments to a function **by pointer**

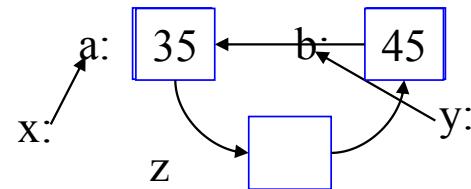
```
passpointer.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 void swap(int *x, int *y) {
5     int z;
6     z = *x;
7     *x = *y;
8     *y = z;
9 }
10
11
12 int main()
13 {
14     int a = 45, b = 35;
15     cout << "Before Swap\n";
16     cout << "a = " << a << ",b = " << b << endl;
17
18     swap(&a, &b);
19
20     cout << "After Swap\n";
21     cout << "a = " << a << ",b = " << b << endl;
22
23     return 0;
24 }
```

before calling:

a: 45

b: 35

calling:



after calling:

a: 35

b: 45

Output:

Before Swap

a = 45,b = 35

After Swap

a = 35,b = 45

How to check in functions in Visual Studio 2019?

Set a breakpoint and Start Debugging (shortcut:F5)

The screenshot shows the Visual Studio 2019 IDE during a debugging session. The code editor displays `testFunction.cpp` with a breakpoint at line 17 (circled in red). The code defines a swap function and a main function that swaps the values of `a` and `b`, then prints them before and after the swap. The Watch 1 window shows the variables `a` and `b` with their current values (45 and 35). The Call Stack window shows the current call stack frame is at line 17 of `testFunction.exe`. The Diagnostic Tools window shows memory usage and CPU usage over time.

You can watch the values of a and b, not x and y, because x and y are out of their scope.

Input variables you want to check in watch window

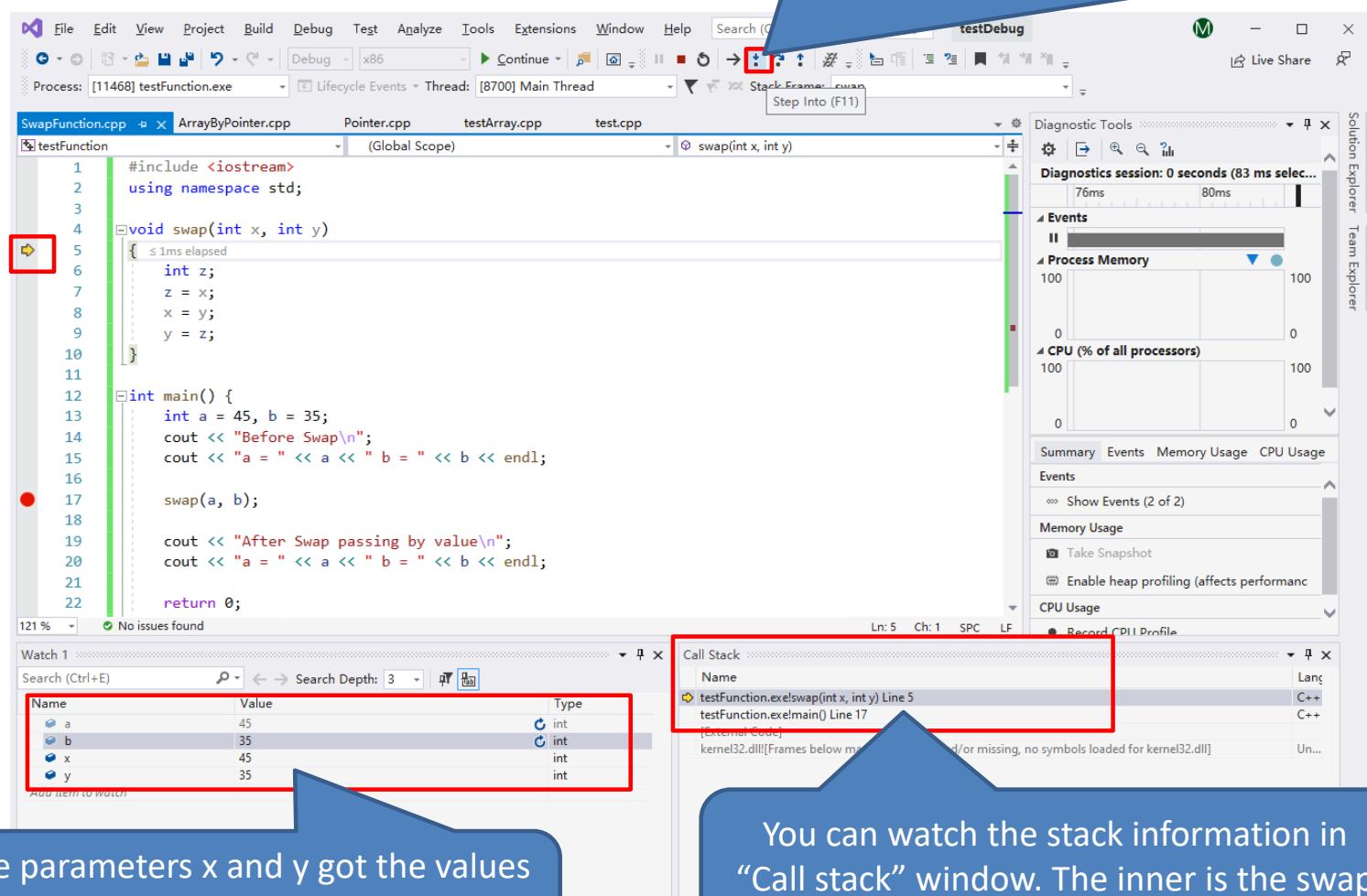
Call Stack

Name	Type
a	int
b	int
x	identifier "x" is undefined
y	identifier "y" is undefined

Watch 1

Name	Value	Type
a	45	int
b	35	int

Press “step into(F11)” to run the program not only step by step but also into the function



The parameters x and y got the values of arguments a and b, and the control flow goes into the function.

You can watch the stack information in “Call stack” window. The inner is the swap function, the outer is the main function.

The screenshot shows a Microsoft Visual Studio IDE interface during a debugging session. The title bar indicates the project is named "testFunction" and the configuration is "testDebug".

Code Editor: The main window displays the file "SwapFunction.cpp" with the following code:

```
#include <iostream>
using namespace std;

void swap(int x, int y)
{
    int z;
    z = x;
    x = y;
    y = z;
} ≤ 1ms elapsed

int main() {
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << endl;

    swap(a, b);

    cout << "After Swap passing by value\n";
    cout << "a = " << a << " b = " << b << endl;

    return 0;
}
```

A red box highlights the first line of the swap function definition. A red circle marks the current breakpoint at the start of the swap function. A red rectangle highlights the variable "x" in the "swap" function's parameter list.

Watch Window: The "Watch 1" window shows the following variable values:

Name	Type	Value
a	int	45
b	int	35
x	int	35
y	int	45
z	int	45

A red box highlights the row for variable "x".

Call Stack: The "Call Stack" window shows the current stack frames:

Name	Language
testFunction.exe!swap(int x, int y) Line 10	C++
testFunction.exe!main() Line 17	C++
[External Code]	Unloaded
kernel32.dll!Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll	Unloaded

Diagnostic Tools: The "Diagnostic Tools" window shows performance metrics over time:

- Events:** Shows a timeline with several events occurring.
- Process Memory:** Shows memory usage fluctuating between 0 and 100.
- CPU (% of all processors):** Shows CPU usage fluctuating between 0 and 100.

Below the charts are tabs for "Summary", "Events", "Memory Usage", and "CPU Usage".

When the function runs to the end, the values of x and y are changed.

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) testDebug

Process: [11468] testFunction.exe Lifecycle Events Thread: [8700] Main Thread Stack Frame: main

SwapFunction.cpp ArrayByPointer.cpp Pointer.cpp testArray.cpp test.cpp

testFunction

(Global Scope)

main()

```
1 #include <iostream>
2 using namespace std;
3
4 void swap(int x, int y)
5 {
6     int z;
7     z = x;
8     x = y;
9     y = z;
10 }
11
12 int main() {
13     int a = 45, b = 35;
14     cout << "Before Swap\n";
15     cout << "a = " << a << " b = " << b << endl;
16
17     swap(a, b);
18
19     cout << "After Swap passing by value\n"; 1ms elapsed
20     cout << "a = " << a << " b = " << b << endl;
21
22     return 0;
}
```

121 % No issues found

Ln: 19 Ch: 1 SPC LF

Watch 1

Name	Value	Type
a	45	int
b	35	int
x	35	int
y	45	int
z		int

Call Stack

Name	Language
testFunction.exe!main() Line 19	C++
[External Code]	
kernel32.dll![Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]	Un...

Diagnostic Tools

Diagnostics session: 0 seconds (84 ms selected) 83.2ms

Events

Process Memory

CPU (% of all processors)

Summary Events Memory Usage CPU Usage

Events

Show Events (8 of 8)

Memory Usage

Take Snapshot

Enable heap profiling (affects performance)

CPU Usage

Record CPU Profile

Autos Locals Watch 1 Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Ready Add to Source Control

After the function calling, the control flow returns back to the caller(main function), the values of a and b are not changed due to passing arguments by value.

Passing arguments to a function by pointer

The screenshot shows the Microsoft Visual Studio IDE interface during a debugging session. The main window displays the source code for `testFunction.cpp`. A red box highlights the `swap` function, which takes two integer pointers as parameters and swaps their values. A blue callout points to this function with the text: "Modify the value in the function using its parameter by pointer". Another red box highlights the call to `swap(&a, &b);` in the `main` function, with a blue callout pointing to it and the text: "Call the function passing by address of arguments". A third red box highlights the variable `&a` in the `Watch 1` window, with a blue callout pointing to it and the text: "Check the address of a and b". The `Call Stack` window shows the current stack frame as `testFunction.exe!main() Line 17 [External Code]`.

```
#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
    int z;
    z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << endl;

    swap(&a, &b);

    cout << "After Swap passing by value\n";
    cout << "a = " << a << " b = " << b << endl;

    return 0;
}
```

Watch 1

Name	Value	Type
<code>&a</code>	0x00effa24 (45)	int *
<code>&b</code>	0x00effa18 (35)	int *
<code>x</code>	Identifier 'x' is undefined	
<code>y</code>	Identifier 'y' is undefined	

Call Stack

Name	Lang
testFunction.exe!main() Line 17 [External Code]	C++
kernel32.dll![Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]	Un...

Autos Locals Watch 1 Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Ready Add to Source Control

Press “step into(F11)” to step into the function

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+F) testDebug M - □ × Live Share

Process: [7964] testFunction.exe Lifecycle Events Thread: [23772] Main Thread Stack Frame: swap

SwapFunction.cpp ArrayByPointer.cpp Pointer.cpp testArray.cpp test.cpp (Global Scope)

swap(int *x, int *y)

```
1 #include <iostream>
2 using namespace std;
3
4 void swap(int *x, int *y)
5 {
6     if (1ms elapsed
7         int z;
8         z = *x;
9         *x = *y;
10        *y = z;
11    }
12
13 int main() {
14     int a = 45, b = 35;
15     cout << "Before Swap\n";
16     cout << "a = " << a << " b = " << b << endl;
17
18     swap(&a, &b);
19
20     cout << "After Swap passing by value\n";
21     cout << "a = " << a << " b = " << b << endl;
22
23     return 0;
24 }
```

No issues found

Ln: 5 Ch: 1 SPC LF

Watch 1

Name	Type
↳ &a	int *
↳ &b	int *
↳ x	int *
↳ y	int *
z	int

Call Stack

Name	Lang
testFunction.exe!swap(int *x, int *y) Line 5	C++
testFunction.exe!main() Line 17	C++
[External Code]	
kernel32.dll!Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]	Un...

x points to a and y points to b, which means x and a occupies the same space of memory, y and b occupies another same space of memory

Autos Loc Intermediate Window Output

Ready Add to Source Control

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help

Process: [7964] testFunction.exe Lifecycle Events Thread: [23772] Main Thread Stack Frame: main

SwapFunction.cpp ArrayByPointer.cpp Pointer.cpp testArray.cpp test.cpp

```
#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
    int z;
    z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << endl;

    swap(&a, &b);

    cout << "After Swap passing by value\n";
    cout << "a = " << a << " b = " << b << endl;

    return 0;
}
```

121 % No issues found

Watch 1

Name	Type	Value
&a	int*	0x00effa24 {35}
&b	int*	0x00effa18 {45}
y	int*	0x00effa24 {35}
z	int	45

Call Stack

Name	Lang
testFunction.exe!main() Line 19	C++
[External Code]	
kernel32.dll!Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll	Un...

Diagnostic Tools

Diagnostics session: 0 seconds (56 ms selected)

Events

Process Memory (KB)

CPU (% of all processors)

Summary Events Memory Usage CPU Usage

Events

Memory Usage

Take Snapshot

Enable heap profiling (affects performance)

CPU Usage

Record CPU Profile

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Ready Add to Source Control

After calling the function, x and y are out of its scope, but the exchanging is finished in the function at the memory space in which a and b are occupied.

How to check in functions in CLion?

Set a breakpoint and choose Debug item



Debug 'swapfunction'

The screenshot shows the CLion IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project Tab:** swapfunction / main.cpp
- CMakeLists.txt Tab:** CMakeLists.txt
- Code Editor:** main.cpp
- Breakpoint:** A red circle highlights the breakpoint at line 17, which contains the call to `myswap(a,b);`.
- Output:** swapfunction - main.cpp
- Toolbars:** Database, etc.
- Bottom Bar:** Debug: swapfunction
- Debugger Tab:** Shows Thread-1 and main main.cpp:17.
- Variables Tab:** Shows variables `a = {int} 35` and `b = {int} 45`.
- Watches Tab:** Shows two entries with error messages:
 - `x = -var-create: unable to create variable object`
 - `y = -var-create: unable to create variable object`

You can watch the values of local variable in “Variable” tag, such as the value of a and b.

You can input the name of variable you want to watch in “Watch” tag.

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help swapfunction - main.cpp

swapfunction \ main.cpp

Project CMakeLists.txt main.cpp

```
#include <iostream>
using namespace std;

void myswap(int x, int y)
{
    int z;
    z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 35, b = 45;    a: 35    b: 45
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    myswap(a,b);    a: 35    b: 45

    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

Press "Step Into F7" button to run into the function

Debug: swapfunction

Debugger Console

Frames Variables GDB Memory View Watches

Thread-1 main main.cpp:17

a = {int} 35
b = {int} 45

x = -var-create: unable to create variable object
y = -var-create: unable to create variable object

Run TODO Problems Debug Terminal CMake Messages Event Log

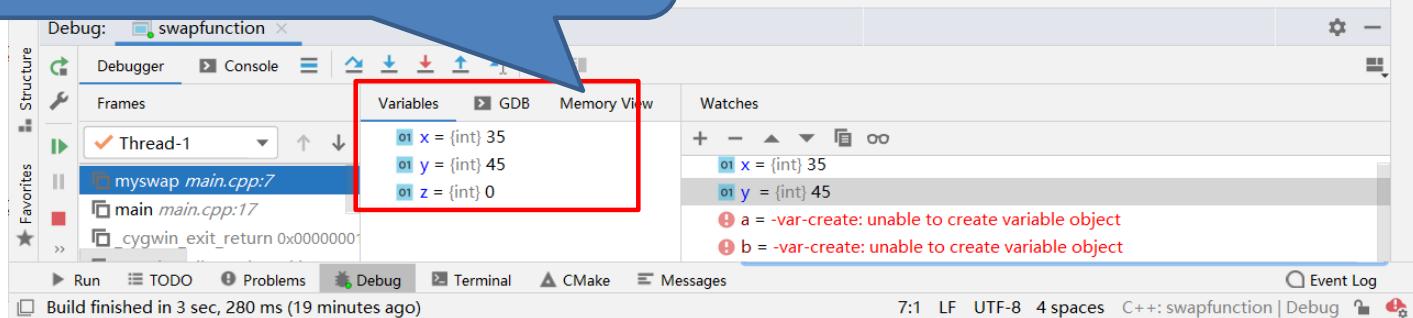
Build finished in 3 sec, 280 ms (2 minutes ago) 17:1 LF UTF-8 4 spaces C++: swapfunction | Debug

```
#include <iostream>
using namespace std;

void myswap(int x, int y) x: 35 y: 45
{
    int z; z: 0
7 → z = x; x: 35 z: 0
    x = y;
    y = z;
}
int main()
{
    int a = 35, b = 45;
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    myswap(a,b);
}
```

The parameters x and y got the values of arguments a and b, and the control flow goes into the function.



The screenshot shows a CLion IDE interface with the following details:

- Project:** swapfunction
- File:** main.cpp
- Editor:** CMakeLists.txt (tab selected), main.cpp (code view). The code defines a swap function and prints values before and after the swap.
- Toolbars:** Standard file operations (File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help).
- Bottom Bar:** Debug: swapfunction, Run, TODO, Problems, Step to the next line in this file, Event Log, UTF-8 4 spaces, C++: swapfunction | Debug.
- Bottom Left:** Structure, Favorites, Debugger (selected), Console, Memory View, GDB, Watches.
- Bottom Middle:** Variables tab (selected) showing local variables: x = (int) 45, y = (int) 35, z = (int) 35. A red box highlights this tab.
- Bottom Right:** Watches tab showing the same variable values.

A red circle highlights the step-over arrow icon at the start of line 10 in the editor. A blue callout bubble at the bottom states: "When the function runs to the end, the values of x and y are changed."

```
#include <iostream>
using namespace std;

void myswap(int x, int y)    x: 45    y: 35
{
    int z;    z: 35
    z = x;
    x = y;    x: 45
    y = z;    y: 35    z: 35
}

int main()
{
    int a = 35, b = 45;
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    myswap(a,b);

    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

When the function runs to the end, the values of x and y are changed.

The screenshot shows the CLion IDE interface with the following details:

- Project Bar:** Shows the project name "swapfunction" and file "main.cpp".
- Code Editor:** Displays the CMakeLists.txt and main.cpp files. The main.cpp code is as follows:

```
#include <iostream>
using namespace std;

void myswap(int x, int y)
{
    int z;
    z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 35, b = 45;    a: 35    b: 45
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    myswap(a,b);    a: 35    b: 45
    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

- Debugger View:** Shows the "Variables" tab with the following content:
- Thread-1
- main main.cpp:19
- a = {int} 35
- b = {int} 45
- Bottom Status Bar:** Shows "Build finished in 3 sec, 280 ms (24 ms)".

After the function calling, the control flow returns back to the caller(main function), the values of a and b are not changed due to passing arguments by value.

Passing arguments to a function by pointer

The screenshot shows the CLion IDE interface with a C++ project named "swapfunction". The main.cpp file contains the following code:

```
#include <iostream>
using namespace std;

void myswap(int *x, int *y)
{
    int z;
    z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 35, b = 45;    a: 35    b: 45
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    myswap(&a, &b);    a: 35    b: 45

    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

The code defines a function `myswap` that swaps the values of two integers passed by pointer. In the `main` function, it swaps the values of `a` and `b`, which are initially 35 and 45 respectively. The output shows the state before and after the swap.

In the bottom right corner of the IDE, there is a blue callout bubble with the text "Watch the address of a and b".

The Watch View in the bottom right pane shows the variables `a` and `b`. The variable `a` is set to 35, and the variable `b` is set to 45. The memory addresses for `&a` and `&b` are highlighted with a red box. The addresses are `0xfffffcbd0` and `0xfffffcbd8` respectively. Below the variables, there are error messages indicating that variables `x` and `v` could not be created.

```
#include <iostream>
using namespace std;

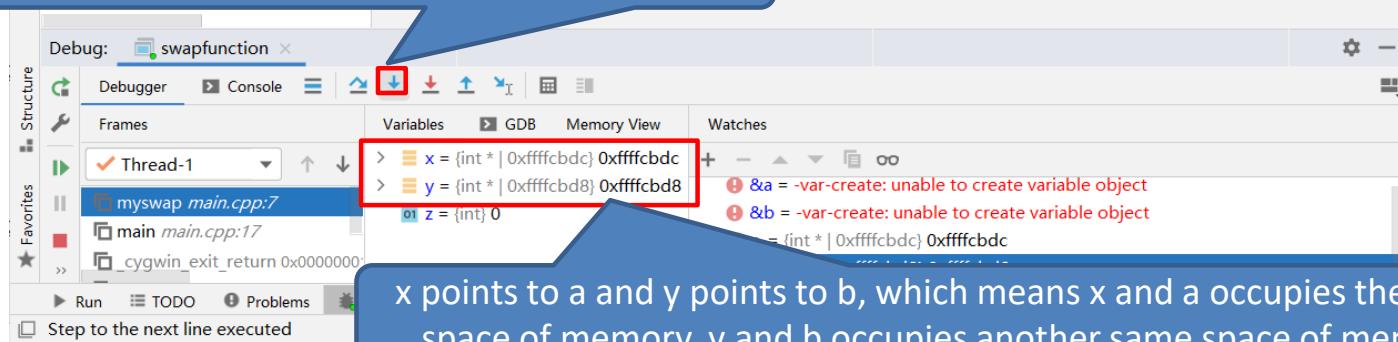
void myswap(int *x, int *y)    x: 0xfffffcdbc    y: 0xfffffcbd8
{
    int z;    z: 0
    z = *x;    x: 0xfffffcdbc    z: 0
    *x = *y;
    *y = z;
}

int main()
{
    int a = 35, b = 45;
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    myswap(&a, &b);

    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;
}
```

Press “Step Into F7” to step into the function



x points to a and y points to b, which means x and a occupies the same space of memory, y and b occupies another same space of memory

The screenshot shows the CLion IDE interface with the following details:

- Project View:** Shows the project structure with files like CMakeLists.txt and main.cpp.
- Code Editor:** Displays the main.cpp file containing a swap function and its usage in the main function. A red circle highlights the step-over arrow icon (next to the step-into icon) on line 19.
- Variables View:** Shows the current variable values:
 - Thread-1: a = {int} 45
 - Thread-1: b = {int} 35A red box highlights these two entries.
- Registers View:** Shows memory locations and their values:
 - &a = {int *} 0xffffcbdc
 - a = {int} 35
 - &b = {int *} 0xffffcbd8
 - b = {int} 45
- Status Bar:** Shows file encoding (LF), character width (4 spaces), and the current configuration (C++: swapfunction | Debug).

After calling the function, x and y are out of its scope, but the exchanging is finished in the function at the memory space in which a and b are occupied.

3. Passing arrays to a function (array name as parameters and arguments)

```
passarray.cpp > main()
1 #include <iostream>
2 using namespace std;
3
4 void sum(int arr1[], int arr2[],int n);
5
6 int main()
7 {
8     int a[5] = {10,20,30,40,50};
9     int b[5] = {1,2,3,4,5};
10
11    cout << "Before calling the function, the contents of a are:\n";
12    for(int i = 0; i < 5; i++)
13        cout << a[i] << " ";
14
15    sum(a,b,5);
16
17    cout << "\nAfter calling the function, the contents of a are:\n";
18    for(int i = 0; i < 5; i++)
19        cout << a[i] << " ";
20    cout << endl;
21
22    return 0;
23 }
24 void sum(int arr1[],int arr2[],int n)
25 {
26     int temp;
27     for(int i = 0; i < n; i++)
28     {
29         temp = arr1[i] + arr2[i];
30         arr1[i] = temp;
31     }
32 }
```

general
Using array as a parameter
arr1 = &a[0] or arr1 = a
用 arr.size() 也可

```
Before calling the function, the contents of a are:
10 20 30 40 50
After calling the function, the contents of a are:
11 22 33 44 55
```

The values of elements in array a are changed.

3. Passing arrays to a function (pointers as parameters and array name as arguments)

```
passarray2.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 void sum(int *p1, int *p2,int n);
5
6 int main()
7 {
8     int a[5] = {10,20,30,40,50};
9     int b[5] = {1,2,3,4,5};
10
11    cout << "Before calling the function, the contents of a are:\n";
12    for(int i = 0; i < 5; i++)
13        cout << a[i] << " ";
14
15    sum(a,b,5);
16
17    cout << "\nAfter calling the function, the contents of a are:\n";
18    for(int i = 0; i < 5; i++)
19        cout << a[i] << " ";
20    cout << endl;
21
22    return 0;
23 }
24 void sum(int *p1,int *p2,int n)
25 {
26     int temp;
27     for(int i = 0; i < n; i++)
28     {
29         temp = *p1 + *p2;
30         *p1 = temp;
31         p1++;
32         p2++;
33     }
34 }
```

Using pointer as a parameter

p1 = a or p1 = &a[0]

```
Before calling the function, the contents of a are:
10 20 30 40 50
After calling the function, the contents of a are:
11 22 33 44 55
```

The values of elements in array a are changed.

3. Passing arrays to a function

(protect the value of the argument from modifying, please use **const** pointer)

```
passarray3.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 void sum(const int *p1, const int *p2, int n);
5
6 int main()
7 {
8     int a[5] = {10,20,30,40,50};
9     int b[5] = {1,2,3,4,5};
10
11    cout << "Before calling the function, the contents of a are:\n";
12    for(int i = 0; i < 5; i++)
13        cout << a[i] << " ";
14
15    sum(a,b,5);
16
17    cout << "\nAfter calling the function, the contents of a are:\n";
18    for(int i = 0; i < 5; i++)
19        cout << a[i] << " ";
20    cout << endl;
21
22    return 0;
23 }
24 void sum(const int *p1,const int *p2,int n)
25 {
26     int temp;
27     for(int i = 0; i < n; i++)
28     {
29         temp = *p1 + *p2;
30         *p1 = temp; // Error: assignment of read-only location
31         p1++;
32         p2++;
33     }
34 }
```

只能读取，不能修改

recommended to use the **pointer-to-const** form to protect data!!

Using const pointer

as a parameter

The value of array
can not be modified.

passarray3.cpp: In function ‘void sum(const int*, const int*, int)’:
passarray3.cpp:30:13: error: assignment of read-only location “* p1”

30 | *p1 = temp;
| ~~~~~^~~~~~

4. Passing multidimensional array to a function

```
#include <iostream>
using namespace std;

void square(const int arr[][3],int n);
```

```
int main()
{
    int a[2][3] = {
        {1,2,3},{4,5,6}
    };
    square(a,2);

    return 0;
}
```

```
void square(int arr[][3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = arr[i][j];
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

指向行向量的指针

```
void square(int (*p)[3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = *(*(p + i) + j);
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

the same as
 $p[i][j]$

```
void square(const int (*p)[3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = p[i][j];
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

```

void square(const int **p, int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
    {
        temp = p[i][j];
        cout << temp * temp << " ";
    }
    cout << endl;
}

```

If the function definition is like this, can we invoke the function by two-dimensional array name?

二级指针不知道如何移动

Compiling errors in VS code

```

/usr/bin/ld: /tmp/ccIRcptd.o: in function `main':
pass2darray.cpp:(.text+0x52): undefined reference to `square(int const (*) [3], int)'
collect2: error: ld returned 1 exit status

```

二级指针不能访问到二维数组

E0167 argument of type "int (*)[3]" is incompatible with parameter of type "const int **"

C2664 'void square(const int **,int)': cannot convert argument 1 from 'int [2][3]' to 'const int **'

```

error: cannot convert 'int (*)[3]' to 'const int**' ?
68 |     square(arr);
|     ^~~
|     |
|     int (*)[3]

```

Compiling errors in Visual Studio

Compiling errors in CLion

```

note: initializing argument 1 of 'void square(const int**)'
48 | void square(const int **p)
|      ~~~~~^

```

5. Passing C-style string to a function

```
passcstring.cpp > ...
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 void mcopy(char *s,int m);
6
7 int main()
8 {    void mcopy(char *s,int m);
9     char str[81];
10    int m;
11    cout<<"Enter a string:\n";
12    cin.getline(str,80);
13
14    cout<<"Enter m:\n";
15    cin>>m;
16
17    mcopy(str,m);
18
19    cout << str << endl;
20
21    return 0;
22 }
23
24 void mcopy(char *s,int m)
25 {
26     strcpy(s,s+m-1);
27 }
```

You can use **character array** or **pointer-to-char** as a parameter.

Output:

```
Enter a string:
Today is a sunny day.
Enter m:
6
is a sunny day.
```

6. Passing structure to a function

```
#include <stdio.h>
#include <string.h>
struct student
{
    int id;
    char name[20];
    float score;
};

void PrintStudent(struct student record);

int main()
{
    struct student record;

    record.id = 1;
    strcpy_s(record.name, "Raju");
    record.score = 86.5;

    PrintStudent(record);
    return 0;
}

void PrintStudent(struct student record)
{
    printf("Id is: %d\n", record.id);
    printf("Name is: %s\n", record.name);
    printf("Score is: %.1f\n", record.score);
}
```

Passing structure to
function by value

```
#pragma once
struct student
{
    int id;
    char name[20];
    float score;
};

void PrintStudent(struct student* record);
```

Passing structure to
function by pointer

```
#include <iostream>
#include <string.h>
#include "student.h"

void PrintStudent(struct student* record)
{
    printf("Id is: %d\n", record->id);
    printf("Name is: %s\n", record->name);
    printf("Score is: %.1f\n", record->score);
}
```

student.cpp
函数体

```
#include <iostream>
#include <string.h>
#include "student.h"

int main()
{
    struct student record;

    record.id = 1;
    strcpy_s(record.name, "Raju");
    record.score = 86.5;

    PrintStudent(&record);
    return 0;
}
```

main.cpp

7. Return an array from a function

```
returnarray.cpp > main()
1 #include <iostream>
2 using namespace std;
3
4 int * fun()
5 {
6     int arr[5];
7     for(int i = 0; i < 5; i++)
8         arr[i] = (i + 1) * 10;
9
10    return arr;
11 }
12
13 int main()
14 {
15     int *p = fun();
16
17     for(int i = 0; i < 5; i++)
18         cout << p[i] << " ";
19     cout << endl;
20
21     return 0;
22 }
```

arr is a local variable

Return the address of a local variable is not right.

```
returnarray.cpp: In function ‘int* fun()’:
```

```
returnarray.cpp:10:12: warning: address of local variable ‘arr’ returned [-Wreturn-local-addr]
```

```
10 |     return arr;
|     ^~~~
```

```
returnarray.cpp:6:9: note: declared here
```

```
6 |     int arr[5];
|     ^~~~
```

Segmentation fault (core dumped)

You can not run the program.

Return an array from function by using dynamically allocated array:

Three correct ways of returning array:

Return an array

1.Using dynamically allocated array

2. Using static array

3.Using structure

不可反局部变量指针

release the memory in caller

```
returnarray.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 int * fun()
5 {
6     int *arr = new int[5];
7     for(int i = 0; i < 5; i++)
8         arr[i] = (i + 1) * 10;
9
10    return arr;
11 }
12
13 int main()
14 {
15     int *p = fun();
16
17     for(int i = 0; i < 5; i++)
18         cout << p[i] << " ";
19     cout << endl;
20     delete []p;
21
22     return 0;
23 }
```

arr is a dynamically allocated array

☆☆

Output:

10 20 30 40 50

Return an array from function by using a static array:

```
#include <iostream>
using namespace std;

int * fun()
{
    static int arr[5];
    for(int i = 0; i < 5; i++)
        arr[i] = (i + 1) * 10;

    return arr;
}

int main()
{
    int *p = fun();

    for(int i = 0; i < 5; i++)
        cout << p[i] << " ";
    cout << endl;

    return 0;
}
```

arr is a static array

return the static arr

Output:

10 20 30 40 50

Return an array from function by **using a structure:**

```
#include <iostream>
using namespace std;

struct arrWrap
{
    int arr[5];
};

struct arrWrap fun()
{
    struct arrWrap x;
    for(int i = 0; i < 5; i++)
        x.arr[i] = (i + 1) * 10;
    return x;
}

int main()
{
    struct arrWrap x = fun();

    for(int i = 0; i < 5; i++)
        cout << x.arr[i] << " ";
    cout << endl;

    return 0;
}
```

arr is a member of a structure

Return the structure

Output:

```
10 20 30 40 50
```

7. Return pointer from a function

```
returnpointer.cpp > ...
30
31 #include <iostream>
32 #include <cstring>
33 using namespace std;
34
35 char *match(char *s, char ch)
36 {
37     while(*s != '\0')
38         if(*s == ch)
39             return(s);
40         else
41             s++;
42     return(NULL);
43 }
44
45 int main()
46 {
47     char ch, str[81], *p = NULL;
48     cout<<"Please input a string:\n";
49     cin.getline(str,80);
50     cout << "Please input a character:\n";
51     ch = getchar( );
52
53     if( ( p = match(str, ch) ) != NULL )
54         cout<< p << endl;
55     else
56         cout << "Not Found\n";
57 }
```

You can return the
parameter pointer

```
Please input a string:
Enjoy the holiday.
Please input a character:
h
he holiday.
```

```
Please input a string:
Class is over.
Please input a character:
m
Not Found
```

C++ program to swap two numbers using pass **by reference**

```
passreference.cpp > main()
1 #include <iostream>
2 using namespace std;
3
4 void swap(int &x, int &y)
5 {
6     int z;
7     z = x;
8     x = y;
9     y = z;
10 }
11
12 int main()
13 {
14     int a = 45, b = 35;
15     cout << "Before Swap\n";
16     cout << "a = " << a << ",b = " << b << endl;
17
18     swap(a,b);
19
20     cout << "After Swap(passing by reference)\n";
21     cout << "a = " << a << ",b = " << b << endl;
22
23     return 0;
24 }
```

output:

```
Before Swap
a = 45,b = 35
After Swap(passing by reference)
a = 35,b = 45
```

C++ program to demonstrate 6 differences between pointer and reference.

Use references when you can, and pointers when you have to.

```
#include <iostream>
using namespace std;
struct demo
{
    int a;
};

int main()
{
    int x = 5;
    int y = 6;
    demo d;

    int *p;
    p = &x; //1. Pointer reinitialization allowed
    p = &y;

    int &r = x;
    // &r = y; //2.Compile Error
    r = y; //2. x value becomes 6

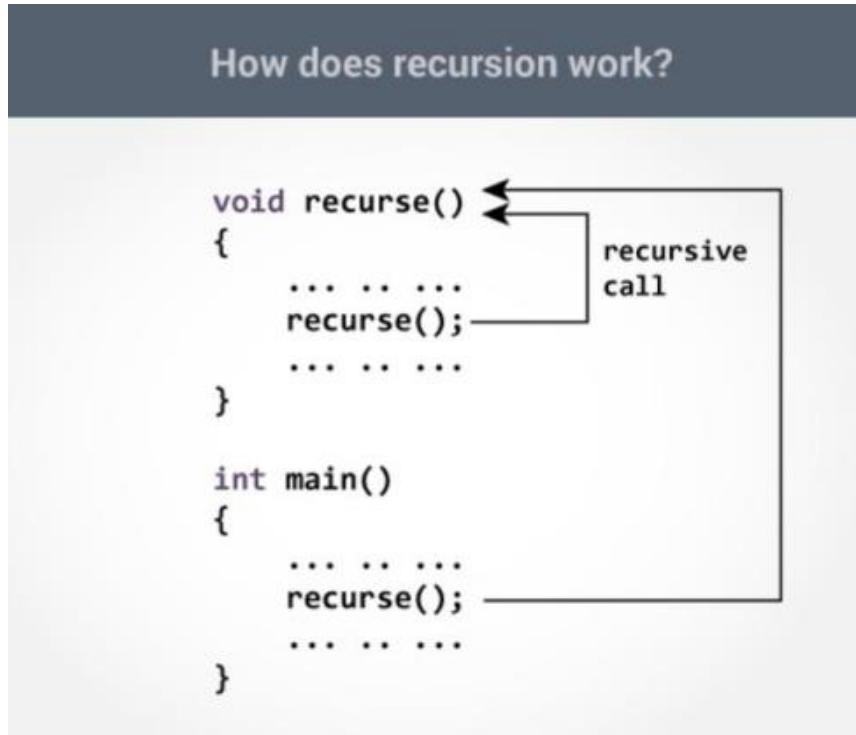
    p = NULL;
    // &r = NULL; //3.Compile error
    p++; //3.Points to next memory location
    r++; //3. x value becomes 7;
    cout << &p << " " << &x << endl; //4.Different address
    cout << &r << " " << &x << endl; //4.Same address

    demo *q = &d;
    demo &qq = d;
    q->a = 8;
    //q.a = 8; //5.Compile Error
    qq.a = 8;
    //qq->a = 8; //5. Compile Error

    cout << p << endl; //6.Prints the address
    cout << r << endl; //6.Prints the value of x
    return 0;
}
```

2.3 Recursive function

A function that **calls itself** is known as **recursive function**. And, this technique is known as **recursion**.



Recursion is used to solve various mathematical problems by dividing it into smaller problems.

Example: compute factorial with recursive function

Compute factorial of a number Factorial of n = 1*2*3...*n

```
recursivefunction.cpp > ...
1 #include <iostream>
2 using namespace std;
3
4 long factorial(int n);
5
6 int main()
7 {
8     long fract;
9     int value;
10    while(true)
11    {
12        cout << "Enter a positive integer:";
13        cin >> value;
14        if(value < 0)
15            cout << "The input must be greater than 0!\n";
16        else
17            break;
18    }
19    fract = factorial(value);
20    cout << "Factorial of " << value << " = " << fract << endl;
21
22    return 0;
23 }
24
25 long factorial(int n)
26 {
27     if(n == 1)
28         return 1;
29     else
30         return n * factorial(n-1);
31 }
```

base condition

- Factorial function: $f(n) = n * f(n-1)$,
- base condition: if $n \leq 1$ then $f(n) = 1$

return $5 * \text{factorial}(4) = 120$

 └ return $4 * \text{factorial}(3) = 24$

 └ return $3 * \text{factorial}(2) = 6$

 └ return $2 * \text{factorial}(1) = 2$

 └ return $1 * \text{factorial}(0) = 1$

Calling itself until the function reaches to the **base condition!**

Output:

```
Enter a positive integer:-4
The input must be greater than 0!
Enter a positive integer:5
Factorial of 5 = 120
```

Direct recursion vs indirect recursion

Direct recursion: When function calls itself, it is called direct recursion

```
#include <iostream>
using namespace std;
int factorial(int n);

int main()
{
    int num = 5;
    cout << factorial(num);

    return 0;
}

int factorial(int n)
{
    if(n < 1)
        return (1);
    else
        return (n * factorial( n: n-1));
}
```

Direct Recursion

Indirect recursion: When function calls another function and that function calls the calling function, then this is called indirect recursion.

```
#include <iostream>
using namespace std;

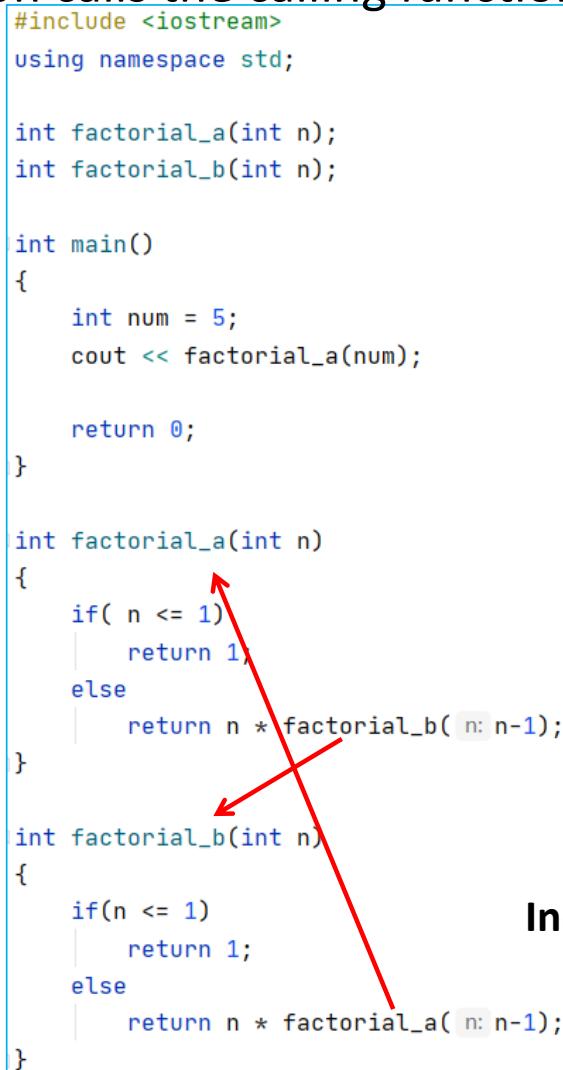
int factorial_a(int n);
int factorial_b(int n);

int main()
{
    int num = 5;
    cout << factorial_a(num);

    return 0;
}

int factorial_a(int n)
{
    if( n <= 1)
        return 1;
    else
        return n * factorial_b( n: n-1);
}

int factorial_b(int n)
{
    if(n <= 1)
        return 1;
    else
        return n * factorial_a( n: n-1);
}
```



Indirect Recursion

Disadvantages of Recursion:

- **Recursive programs are generally slower than nonrecursive programs.** Because it needs to make a function call so the program must save all its current state and retrieve them again later. This consumes more time making recursive programs slower.
- **Recursive programs requires more memory to hold intermediate states in a stack.** Non recursive programs don't have any intermediate states, hence they don't require any extra memory.

2.4 Pointers to Functions(Function Pointer)

Declare a pointer to a function:

return_type (*pointername)(parameter lists);

Return type of a function

The address of a function will be stored in the pointer, which indicates that the pointer is pointed to a function. Note () can not be omitted.

Parameters of a function

Example:

函数名是一个指针

int findmax(int, int);

Declaring a function

int (*funptr)(int,int),

Declaring a pointer to a function

funptr = findmax;

Assigning the address of a function to the pointer

int max = funptr(3,5);

Calling the function by the pointer

Example:

Compute the definite integral, suppose
calculate the following definite integrals

$$\int_a^b f(x) dx = (b-a)/2 * (f(a)+f(b))$$

$$\int_0^1 x^2 dx$$

$$\int_1^2 \sin x/x dx$$

```
#include <iostream>
#include <cmath>
using namespace std;
double calc(double (*func)(double), double a, double b);
double f1(double x1);
double f2(double x2);

int main()
{
    double result;
    double (*func)(double); // Declaring a function pointer
    result = calc(f1, a: 0.0, b: 1.0);
    cout<<"1: result= " << result << endl;
    func = f2; // Assigning the address of function f2 to the pointer
    result = calc(func, a: 1.0, b: 2.0);
    cout<<"2: result= " << result << endl;
    return 0;
}
```

function pointer as a parameter 作函数指针

Declaring a function pointer

Calling the function by function name

Assigning the address of function f2 to the pointer

Calling the function by function pointer

$$\int_a^b f(x)dx = (b-a)/2 * (f(a)+f(b))$$

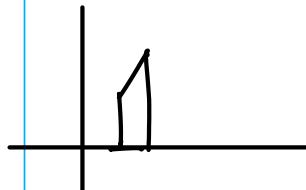
```
double calc ( double (*funp)(double), double a, double b )
{ double z;
  z = (b-a) / 2 * ( (*funp)(a) + (*funp)(b) );
  return ( z );
}
```

```
double f1 ( double x )
{
  return (x * x);
}
```

$$\int_0^1 x^2 dx$$

```
double f2 ( double x )
{
  return (sin(x) / x);
}
```

$$\int_1^2 \frac{\sin x}{x} dx$$



Output:

1: result= 0.5

2: result= 0.64806