# 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>
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);
    cout << "Square root of " << number << " = " << squareRoot;

    return 0;
}
```

header file

sqrt() is a library function

**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

sumfunction.cpp > ...

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

//Declaring a function
int sum(int x, int y);
int main()
{
    int a = 10;
    int b = 20;
    int c;

    //Calling a function
    c = sum(a,b);

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// Defining a function
int sum(int x, int y)
{
    return (x + y);
}
```

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 > ...

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

//Declaring a function
int sum(int x, int y);
int main()
{
    int a = 10;
    int b = 20;
    int c;

    //Calling a function
    c = sum(a,b);

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// Defining a function
int sum(int x, int y)
{
    return (x + y);
}
```

Actual parameters(arguments)

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

Formal parameters

sumfunction.cpp > ...

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

//Declaring a function
int sum(int x, int y);
int main()
{
    int a = 10;
    int b = 20;
    int c;

    //Calling a function
    c = sum(a,b);

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// Defining a function
int sum(int x, int y)
{
    return (x + y);
}
```

30

10

20

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.

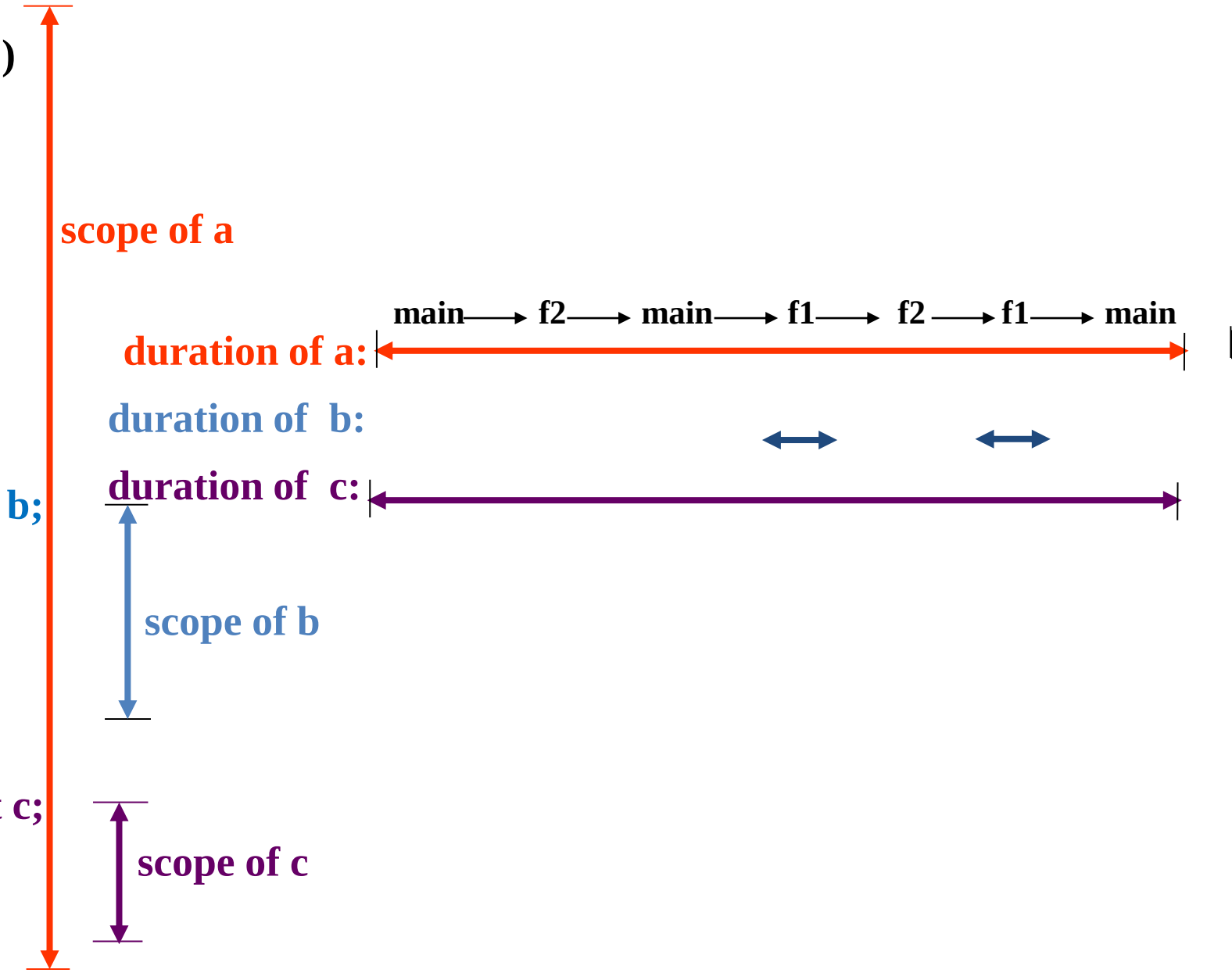
# Scope and duration of variable

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 exits in memory.

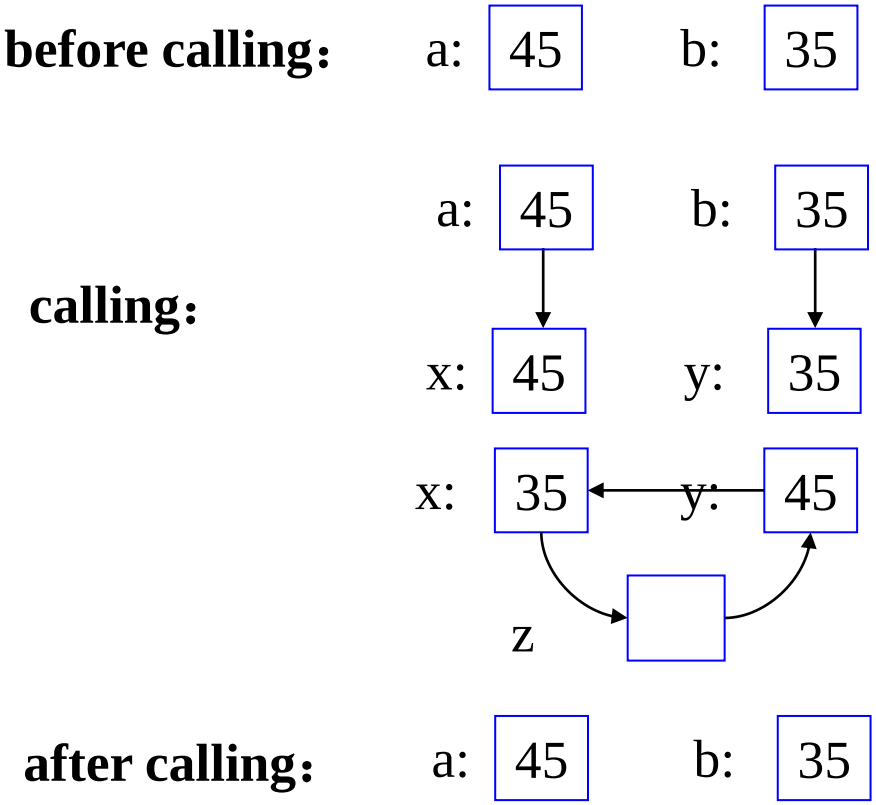
```
int a;
void main( )
{     .......
      .......
        f2;
      .......
        f1;
      .......
}
f1( )
{    auto int b;
      .........
       f2;
       ........
}
f2( )
{   static int c;
      .........
}
```



# 1. Passing arguments to a function by value

passvalue.cpp > ...

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

    return 0;
}
```




Output:

```
Before Swap
a = 45,b = 35
After Swap
a = 45,b = 35
```

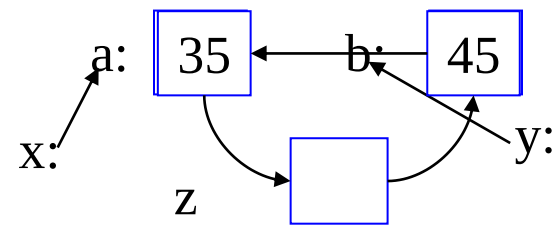
# 2. Passing arguments to a function by pointer

passpointer.cpp > ...

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

    return 0;
}
```

**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)

```
#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;
```

| Name | Value | Type |
|---|---|---|
| a | 45 | int |
| b | 35 | int |
| x | identifier "x" is undefined | |
| y | identifier "y" is undefined | |
| Add item to watch | | |

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

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

```
#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 |
|---|---|---|
| a | 45 | int |
| b | 35 | int |
| x | 45 | int |
| y | 35 | 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 ma... d/or missing, no symbols loaded for kernel32.dll] | Un... |

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.

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

Debug x86 Continue Live Share

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

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

testFunction (Global Scope) swap(int x, int y)

```cpp
#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;
```

121 % No issues found Ln: 10 Ch: 1 SPC LF

Diagnostic Tools

Diagnostics session: 0 seconds (84 ms selec...

83ms 83

Events

Process Memory 100 0 100 0

CPU (% of all processors) 100 0 100 0

Summary Events Memory Usage CPU Usage

Events

Show Events (6 of 6)

Memory Usage

Take Snapshot

Enable heap profiling (affects performanc

CPU Usage

Record CPU Profile

Solution Explorer Team Explorer

Watch 1

Search (Ctrl+E) Search Depth: 3

| Name | Value | Type |
|---|---|---|
| a | 45 | int |
| b | 35 | int |
| x | 35 | int |
| y | 45 | int |
| z | 45 | int |
| Add item to watch | | |

Call Stack

| Name | Lang |
|---|---|
| testFunction.exe!swap(int x, int y) Line 10 | 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... |

mmand Window Immediate Window Output

Ready Add to Source Control

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

Debug x86 Continue Live Share

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

Diagnostic Tools

Diagnostics session: 0 seconds (84 ms selec...

83.2ms

Events

Process Memory 100 0 100 0

CPU (% of all processors) 100 0 100 0

Summary Events Memory Usage CPU Usage

Events

Show Events (8 of 8)

Memory Usage

Take Snapshot

Enable heap profiling (affects performanc

CPU Usage

Record CPU Profile

Solution Explorer Team Explorer

Watch 1

Search (Ctrl+E) Search Depth: 3

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

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... |

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

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

Ready Add to Source Control

# Passing arguments to a function by pointer

```
#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;
}
```

Modify the value in the function using its parameter by pointer

Call the function passing by address of arguments

Watch 1

| Name | Value | Type |
|---|---|---|
| &a | 0x00effa24 {45} | int * |
| &b | 0x00effa18 {35} | int * |
| x | identifier "x" is undefined | |
| y | identifier "y" is undefined | |

Check the address of a and b

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

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctr testDebug

Debug x86 Continue 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

testFunction (Global Scope) swap(int * x, int * y)

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int *x, int *y)
5  {  ≤ 1ms elapsed
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";
20     cout << "a = " << a << " b = " << b << endl;
21
22     return 0;
```

121 % No issues found Ln: 5 Ch: 1 SPC LF

Diagnostic Tools

Diagnostics session: 0 seconds (54 ms selec...

52ms

Events

Process Memory (KB) 928 0

CPU (% of all processors) 100 0

Summary Events Memory Usage CPU Usage

Events

Show Events (2 of 2)

Memory Usage

Take Snapshot

Enable heap profiling (affects performanc

CPU Usage

Record CPU Profile

Solution Explorer Team Explorer

Watch 1

Search (Ctrl+E) Search Depth: 3

| Name | Value | Type |
|---|---|---|
| &a | 0x00effa24 {45} | int * |
| &b | 0x00effa18 {35} | int * |
| x | 0x00effa24 {45} | int * |
| y | 0x00effa18 {35} | int * |
| z | 16061101 | int |

Add item to watch

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... |

Autos Loc ... mediate Window Output

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

Ready Add to Source Control

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

Debug x86 Continue Live Share

Process: [7964] testFunction.exe Lifecycle Events Thread: [23772] 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

Diagnostic Tools

Diagnostics session: 0 seconds (56 ms selec...

54ms 55ms

Events

Process Memory (KB) 928 0 928 0

CPU (% of all processors) 100 0 100 0

Summary Events Memory Usage CPU Usage

Events: Show Events (8 of 8)

Memory Usage: Take Snapshot; Enable heap profiling (affects performanc

CPU Usage: Record CPU Profile

Watch 1

Search (Ctrl+E) Search Depth: 3

| Name | Value | Type |
|---|---|---|
| &a | 0x00effa24 {35} | int * |
| &b | 0x00effa18 {45} | int * |
| x | [illegible] | int * |
| y | [illegible] | int * |
| z | | int |

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... |

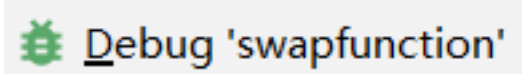
After calling the function, x and y are out of its scope, but the exchanging is finished in the function at the memory space in which a and b are occupied.

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

Ready Add to Source Control

# How to check in functions in CLion?

Set a breakpoint and choose Debug item




```
#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;
}
```

Debug: swapfunction

Variables: a = {int} 35, b = {int} 45

Watches: 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

swapfunction | Debug

Project

swapfunction D:\CLion Project\2

External Libraries

Scratches and Consoles

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

Thread-1

main main.cpp:17

_cygwin_exit_return 0x0000000

_cygtls::call2(unsigned int (*)(v

Variables GDB Memory View

a = {int} 35

b = {int} 45

Watches

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

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

swapfunction 〉 main.cpp

swapfunction | Debug

Project

swapfunction D:\CLion Project\2

External Libraries

Scratches and Consoles

CMakeLists.txt main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  void myswap(int x, int y)   x: 35    y: 45
5  {
6      int z;   z: 0
7      z = x;   x: 35    z: 0
8      x = y;
9      y = z;
10 }
11 int main()
12 {
13     int a = 35, b = 45;
14     cout << "Before Swap\n";
15     cout << "a = " << a << ",b = " << b << endl;
16
17     myswap(a,b);
18
```

Swap\n";

<< a << ",b = " << b << endl;

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

Debug: swapfunction

Debugger Console

Frames

Thread-1

myswap *main.cpp:7*

main *main.cpp:17*

_cygwin_exit_return 0x0000000

Variables GDB Memory View

x = {int} 35

y = {int} 45

z = {int} 0

Watches

x = {int} 35

y = {int} 45

a = -var-create: unable to create variable object

b = -var-create: unable to create variable object

Run TODO Problems Debug Terminal CMake Messages Event Log

Build finished in 3 sec, 280 ms (19 minutes ago)

7:1 LF UTF-8 4 spaces C++: swapfunction | Debug

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

swapfunction | main.cpp

swapfunction | Debug

Project

swapfunction D:\CLion Project\2

External Libraries

Scratches and Consoles

CMakeLists.txt main.cpp

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

myswap

Debug: swapfunction

Debugger Console

Frames

Thread-1

myswap *main.cpp:10*

main *main.cpp:17*

_cygwin_exit_return 0x0000000

Variables GDB Memory View

x = {int} 45

y = {int} 35

z = {int} 35

Watches

x = {int} 45

y = {int} 35

a = -var-create: unable to create variable object

b = -var-create: unable to create variable object

Run TODO Prob

Event Log

Step to the next line in this f

UTF-8 4 spaces C++: swapfunction | Debug

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

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

swapfunction main.cpp swapfunction | Debug

Project swapfunction D:\CLion Project\2C External Libraries Scratches and Consoles

CMakeLists.txt main.cpp

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

main

Debug: swapfunction

Debugger Console

Frames: Thread-1; main *main.cpp:19*; _cygwin_exit_return 0x0000000; _cygtls::call2(unsigned int (*)(v

Variables: a = {int} 35; b = {int} 45

GDB Memory View

Watches: x = -var-create: unable to create variable object; y = -var-create: unable to create variable object; a = {int} 35

Run TODO Problems Event Log

Build finished in 3 sec, 280 ms (24 n

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

# Passing arguments to a function by pointer

```
#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;
}
```

Variables:

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

Watches:

```
&a = {int * | 0xffffcbdc} 0xffffcbdc
&b = {int * | 0xffffcbd8} 0xffffcbd8
x = -var-create: unable to create variable object
y = -var-create: unable to create variable object
```

Watch the address of a and b

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

swapfunction | main.cpp

swapfunction | Debug

Project

swapfunction D:\CLion Project\20

External Libraries

Scratches and Consoles

CMakeLists.txt | main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  void myswap(int *x, int *y)   x: 0xffffcbdc    y: 0xffffcbd8
5  {
6      int z;   z: 0
7      z = *x;   x: 0xffffcbdc    z: 0
8      *x = *y;
9      *y = z;
10 }
11 int main()
12 {
13     int a = 35, b = 45;
14     cout << "Before Swap\n";
15     cout << "a = " << a << ",b = " << b << endl;
16
17     myswap(&a, &b);
18
19     cout << "After Swap\n";
20     cout << "a = " << a << ",b = " << b << endl;
```

Press “Step Into F7” to step into the function

Debug: swapfunction

Debugger | Console

Frames | Variables | GDB | Memory View | Watches

Thread-1

myswap *main.cpp:7*

main *main.cpp:17*

_cygwin_exit_return 0x0000000

x = {int * | 0xffffcbdc} 0xffffcbdc

y = {int * | 0xffffcbd8} 0xffffcbd8

z = {int} 0

&a = -var-create: unable to create variable object

&b = -var-create: unable to create variable object

= {int * | 0xffffcbdc} 0xffffcbdc

Run | TODO | Problems

Step to the next line executed

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

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

swapfunction | main.cpp

swapfunction | Debug

Project

swapfunction D:\CLion Project\2

External Libraries

Scratches and Consoles

CMakeLists.txt | main.cpp

```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: 45    b: 35
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

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

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

    return 0;
}
```

main

Debug: swapfunction

Debugger | Console

Frames | Variables | GDB | Memory View | Watches

Thread-1

a = {int} 45

b = {int} 35

&a = {int * | 0xffffcbdc} 0xffffcbdc

bd8} 0xffffcbd8

ble to create variable object

ble to create variable object

Event Log

LF UTF-8 4 spaces 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()

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

void sum(int arr1[], int arr2[],int n);

int main()
{
    int a[5] = {10,20,30,40,50};
    int b[5] = {1,2,3,4,5};

    cout << "Before calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";

    sum(a,b,5);

    cout << "\nAfter calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";
    cout << endl;

    return 0;
}
void sum(int arr1[],int arr2[],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
    {
        temp = arr1[i] + arr2[i];
        arr1[i] = temp;
    }
}
```

Using array as a parameter

**arr1 = &a[0]** or **arr1 = a**

```
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 > ...

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

void sum(int *p1, int *p2,int n);

int main()
{
    int a[5] = {10,20,30,40,50};
    int b[5] = {1,2,3,4,5};

    cout << "Before calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";

    sum(a,b,5);

    cout << "\nAfter calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";
    cout << endl;

    return 0;
}
void sum(int *p1,int *p2,int n)
{
    int temp;
    for(int i = 0; i < n; i++)
    {
        temp = *p1 + *p2;
        *p1 = temp;
        p1++;
        p2++;
    }
}
```

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 > ...

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

void sum(const int *p1, const int *p2,int n);

int main()
{
    int a[5] = {10,20,30,40,50};
    int b[5] = {1,2,3,4,5};

    cout << "Before calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";

    sum(a,b,5);

    cout << "\nAfter calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";
    cout << endl;

    return 0;
}
void sum(const int *p1,const int *p2,int n)
{
    int temp;
    for(int i = 0; i < n; i++)
    {
        temp = *p1 + *p2;
        *p1 = temp;
        p1++;
        p2++;
    }
}
```

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 (*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(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;
}
```

```
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;
}
```

```cpp
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?

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
```

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 **'

Compiling errors in Visual Studio

```
error: cannot convert 'int (*)[3]' to 'const int**'
   68 |     square(arr);
      |            ^~~
      |            |
      |            int (*)[3]
note:   initializing argument 1 of 'void square(const int**)'
   48 | void square(const int **p)
      |             ~~~~~~~~~~~~~^
```

Compiling errors in CLion

# 5. Passing C-style string to a function

passcstring.cpp > ...

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

void mcopy(char *s,int m);

int main()
{   void mcopy(char *s,int m);
     char str[81];
     int m;
     cout<<"Enter a string:\n";
     cin.getline(str,80);

     cout<<"Enter m:\n";
     cin>>m;

    mcopy(str,m);

    cout << str << endl;

     return 0;
}

void mcopy(char *s,int m)
{
   strcpy(s,s+m-1);
}
```

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

```c
#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

student.h

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

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

Passing structure to function by pointer

student.cpp

```c
#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);
}
```

main.cpp

```c
#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;
}
```

# 7. Return an array from a function

returnarray.cpp > main()

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

int * fun()
{
    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 local variable

Return the address of a local variable is not right.

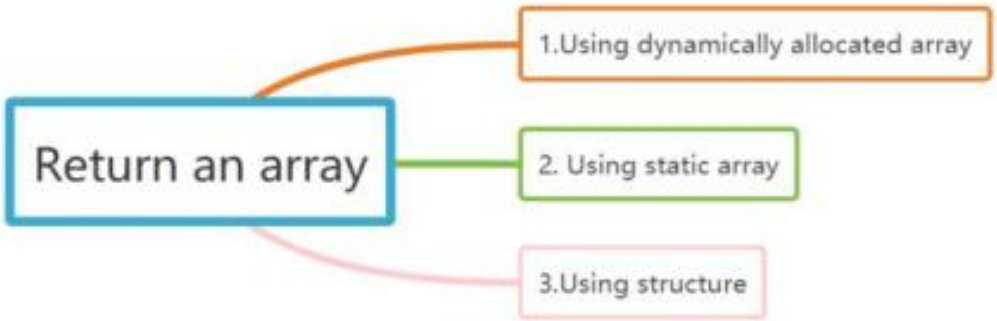
```
returnarray.cpp: In function 'int* fun()':
returnarray.cpp:10:12: warning: address of local variable 'arr' returned [-Wreturn-local-addr]
   10 |     return arr;
      |            ^~~
returnarray.cpp:6:9: note: declared here
    6 |     int arr[5];
      |         ^~~
```

```
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:




returnarray.cpp > ...

```cpp
#include <iostream>
using namespac

int * fun()
{
    int *arr = new int[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;
    delete []p;

    return 0;
}
```

**arr** is a dynamically allocated array

release the memory in caller

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 > ...

```cpp
*/

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

char *match(char *s, char ch)
{    while(*s != '\0')
        if(*s == ch)
            return(s);
        else
            s++;
     return(NULL);
}

int main()
{  char ch, str[81], *p = NULL;
    cout<<"Please input a string:\n";
    cin.getline(str,80);
    cout << "Please input a character:\n";
    ch = getchar( );

    if( ( p = match(str, ch) ) != NULL )
        cout<< p <<endl;
    else
        cout << "Not Found\n";

    return 0;
}
```

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()

```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 reference)\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

output:

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

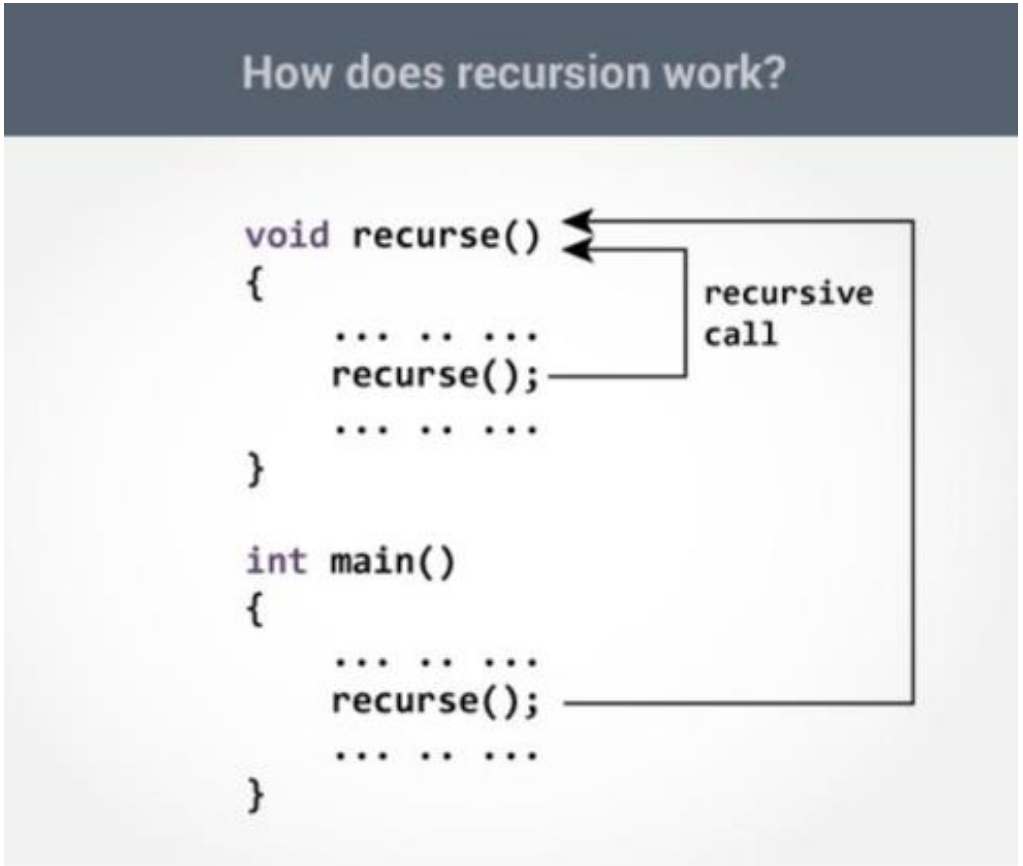
C++ program to demonstrate 6 differences between pointer and reference.

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;
    p = &y;   //1. Pointer reinitialization allowed

    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 > ...

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

long fractorial(int n);

int main()
{
    long fract;
    int value;
    while(true)
    {
        cout << "Enter a positive integer:";
        cin >> value;
        if(value < 0)
            cout << "The input must be greater than 0!\n";
        else
            break;
    }
    fract = fractorial(value);
    cout << "Factorial of " << value << " = " << fract << endl;

    return 0;
}

long fractorial(int n)
{
    if(n == 1)
        return 1;
    else
        return n * fractorial(n-1);
}
```

base condition

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

return 5 * factorial(4) = 120

└ return 4 * factorial(3) = 24

└ return 3 * factorial(2) = 6

└ return 2 * factorial(1) = 2

└ return 1 * 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:

```
int findmax(int, int);
int   (*funptr)(int,int);
funptr = findmax;
int max = funptr(3,5);
```

Declaring a function

Declaring a pointer to a function

Assigning the address of a function to the pointer

Calling the function by the pointer

Example:

Compute the definite integral, suppose calculate the following definite integrals

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

$$\int_0^1 x^2\,dx \qquad \int_1^2 \sin x/x\,dx$$

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

int main()
{
    double result;
    double (*funp)(double);

    result = calc(f1, a: 0.0, b: 1.0);
    cout<<"1: result= " << result << endl;
    funp = f2;
    result = calc(funp, a: 1.0, b: 2.0);
    cout<<"2: result= " << result << endl;

    return 0;
}
```

function pointer as a parameter

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);
}

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

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

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

Output:

```
1: result= 0.5
2: result= 0.64806
```