

Lesson 5.3:

LEARNING OBJECTIVES

By the end of this lesson, you should be able to:

- Explain the two modes of parameter passing

Content Copyright Nanyang Technological University

2

After this lesson, you should be able to:

Explain the two modes of parameter passing

LEARNING OBJECTIVES

By the end of this lesson, you should be able to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable

Explain that the parameter called by reference is a pointer variable

LEARNING OBJECTIVES

By the end of this lesson, you should be able to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable
- Apply call by reference in programming

Content Copyright Nanyang Technological University

4

Apply call by reference in programming

LEARNING OBJECTIVES

By the end of this lesson, you should be able to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable
- Apply call by reference in programming
- Explain double indirection

Explain double indirection

LEARNING OBJECTIVES

By the end of this lesson, you should be able to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable
- Apply call by reference in programming
- Explain double indirection
- Apply pointer dereferencing for swapping operation

Apply pointer dereferencing for swapping operation

PARAMETER PASSING

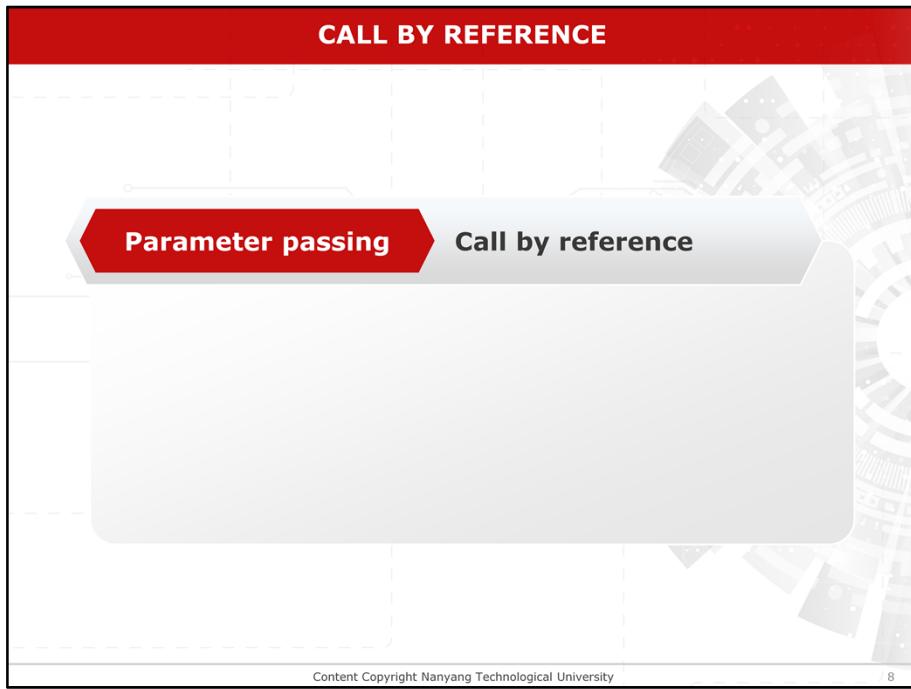
Parameter passing between functions has two modes:

- call by value
- call by reference

Content Copyright Nanyang Technological University

7

Parameter passing between functions can be done either through call by value or call by reference. Call by value has been discussed in the lesson on functions. This lesson will cover on call by reference.



In call by reference,

CALL BY REFERENCE

Parameter passing → **Call by reference**

The parameter in the function holds the **address** of the argument variable

Content Copyright Nanyang Technological University 9

The parameter in the function holds the **addresses** of the argument variables

CALL BY REFERENCE

Parameter passing → Call by reference

The parameter in the function holds the **address** of the argument variable, i.e. the **parameter** is a **pointer variable**.

Content Copyright Nanyang Technological University 10

that is, the parameter is a **pointer variable**.

Therefore, any changes to the values pointed to by the parameters change the arguments. The arguments must be the addresses of variables that are local to the calling function.

CALL BY REFERENCE

- In a function call, the **arguments** must be **pointers** (or using address operator as the prefix)

Content Copyright Nanyang Technological University

11

In a function call, the **arguments** must be **pointers** or using address operator as the prefix.

CALL BY REFERENCE

- In a function call, the **arguments** must be **pointers** (or using address operator as the prefix)
- In the parameter declaration list of the function header, the **parameters** must be prefixed by the **indirection operator (*)**

Content Copyright Nanyang Technological University

12

In the parameter declaration list of the function header, the **parameters** must be prefixed by the **indirection operator**.

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
```



Content Copyright Nanyang Technological University

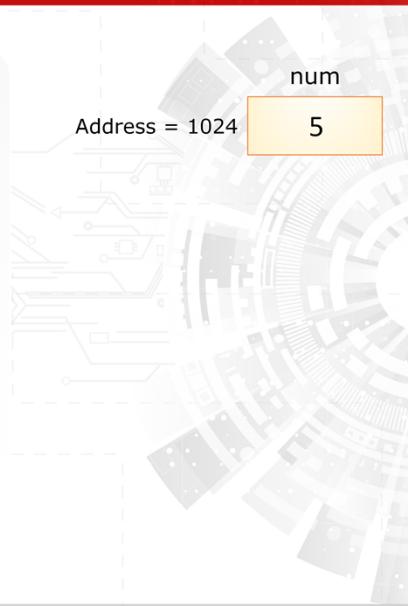
13

Let us look at this program that uses call by reference and go through it step by step.

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

14

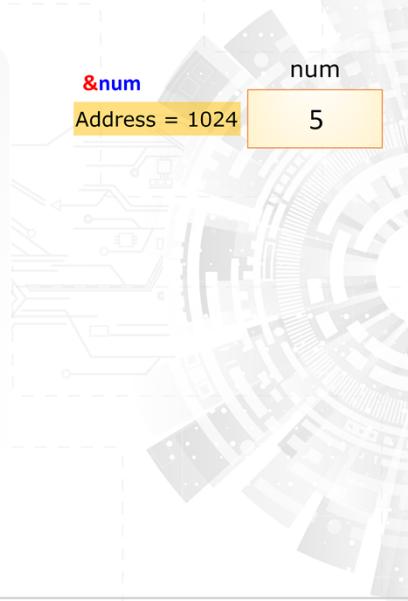
Step 1: the variable **num** is initially assigned with a value 5 in **main()**.

[\[+animation\]](#)

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
```



Content Copyright Nanyang Technological University

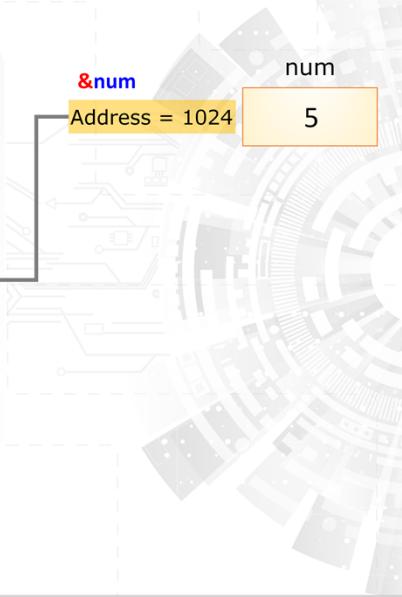
15

Step 2: The address of the variable **num** is then

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

16

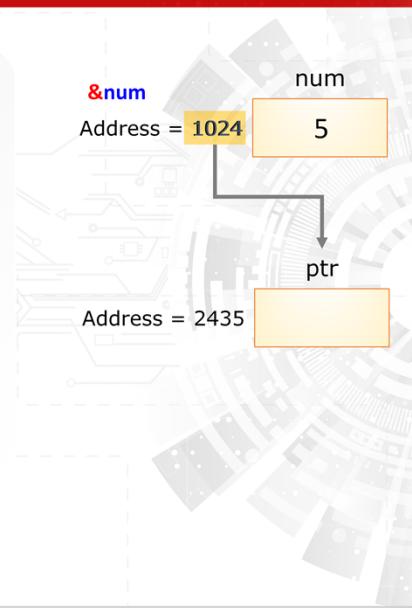
passed as an argument to the function `add2()`

[+animation]

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

17

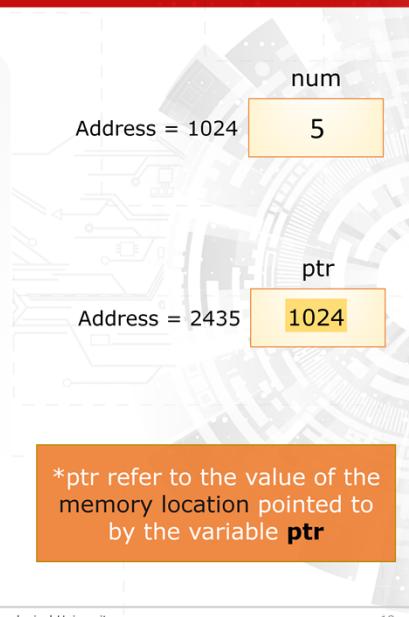
and stored in the parameter **ptr** in the function in step 3.

[\[+animation\]](#)

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

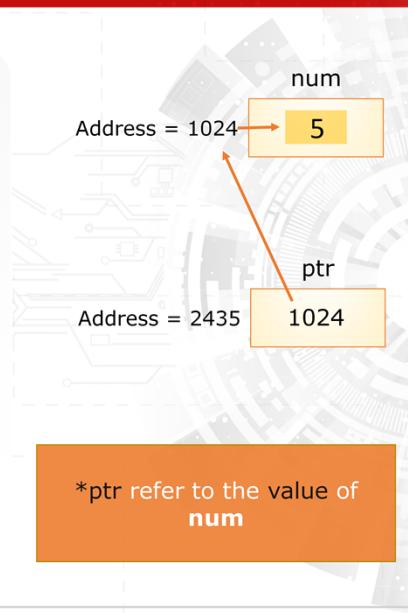
18

Step 4: In the function **add2()**, ***ptr** refer to the value of the memory location pointed to by the variable **ptr**

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

19

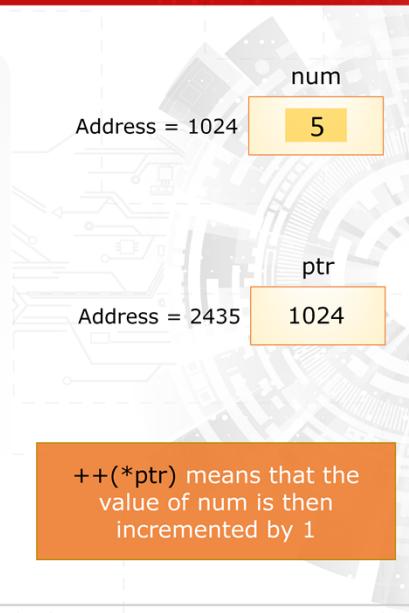
That it **num**

[+animation]

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

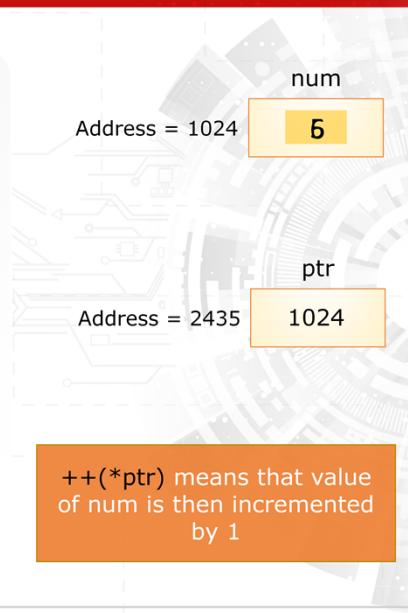
20

++(*ptr) means that the value of num is then incremented by 1

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

21

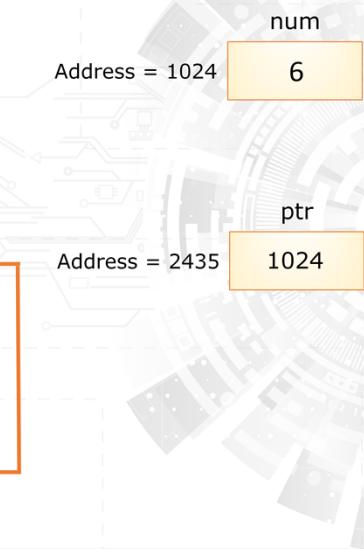
the value stored in the variable **num** increased by 1 from 5 to 6.

[+animation]

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

22

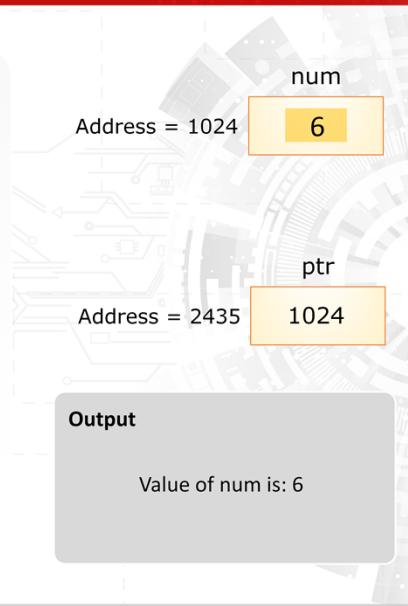
When the function ends, the control is then returned to the calling **main()** function.

[+animation]

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

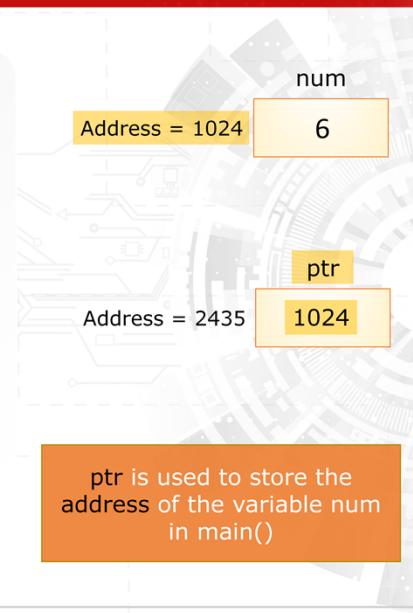
23

Therefore, when **num** is printed, the value **6** is displayed on the screen.

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

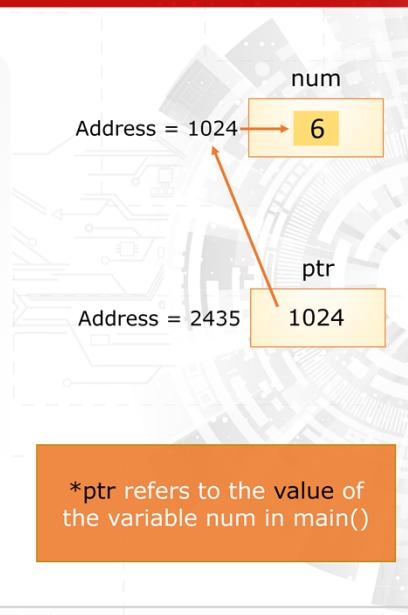
24

In this example, note that the parameter variable **ptr** in **add2()** is used to store the address of the variable **num** in **main()**.

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

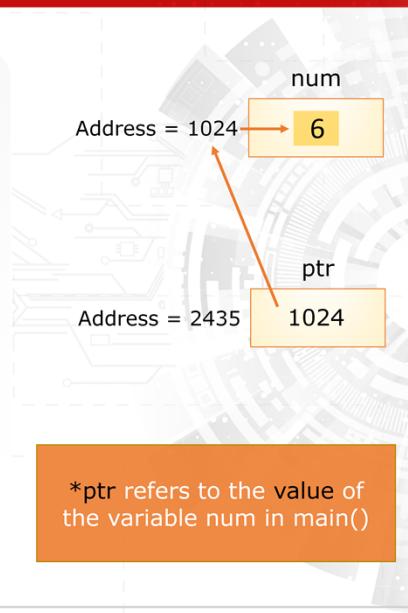
25

*ptr refers to the value of the variable num in the main() function

CALL BY REFERENCE – EXAMPLE 1

```
#include <stdio.h>
void add2(int *ptr);
int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

    return 0;
}
void add2(int *ptr)
{
    ++(*ptr);
}
```



Content Copyright Nanyang Technological University

26

After passing the variable address of **num** into the parameter variable **ptr**, all the operations on **ptr** in the function **add2()** will update the content of the variable **num** indirectly. This example shows how call by reference works.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Content Copyright Nanyang Technological University

27

Let's look at another example for call by reference.

In this example, there are 3 function definitions:

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Content Copyright Nanyang Technological University

28

function1

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Content Copyright Nanyang Technological University

29

function2

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Content Copyright Nanyang Technological University

30

function3

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Call by value

Content Copyright Nanyang Technological University

31

the parameters **a**, **c** and **h** are passed into the functions using call by value,

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Call by reference

Content Copyright Nanyang Technological University

32

whereas the parameters **b**, **d** and **k** are passed into the functions using call by reference

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

Call by reference:
Addresses, not actual value

Content Copyright Nanyang Technological University

33

Addresses are passed to the functions instead of actual values. In fact, the memory contents of these parameters contain the address of the variable **y** in the **main()** function. Any changes to the dereferenced pointers such as ***b**, ***d** and ***k** refer indirectly to the changes to the contents stored in the memory location of the variable **y**.

[show animation]

We will look into each part of the coding in detail.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);

int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}

void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}

void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}

void function3(int h, int *k) {
    *k = *k - h;
}
```

Content Copyright Nanyang Technological University

34

Let's start with the main() function.

CALL BY REFERENCE – EXAMPLE 2

```
int main() {  
    int x, y;  
    x = 5; y = 5;  
    function1(x, &y);  
    return 0;  
}
```

Address = 2002

x

Address = 2048

y

Content Copyright Nanyang Technological University

35

When the program starts execution, in the **main()** function, memory locations are allocated for the variables **x** and **y** accordingly.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
int main() {  
    int x, y;  
    x = 5; y = 5;  
    function1(x, &y);  
    return 0;  
}
```

Address = 2002

x

5

Address = 2048

y

5

Content Copyright Nanyang Technological University

36

The two variables are assigned with the value of 5.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
```

Address = 2002	x 5
Address = 2048	y 5

Content Copyright Nanyang Technological University

37

Therefore, the memory locations of the variables **x** and **y** store the value of 5 directly.

Let's track the changes in the parameters to observe the effect of call by reference as we run through the key steps in codes.

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-

Content Copyright Nanyang Technological University

38

The declared values of x and y are 5.

CALL BY REFERENCE – EXAMPLE 2

```
int main() {  
    int x, y;  
    x = 5; y = 5;  
    function1(x, &y);  
    return 0;  
}
```

Address = 2002	x 5
Address = 2048	y 5

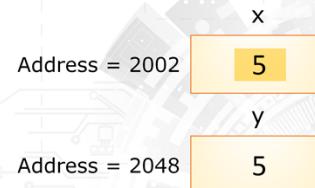
Content Copyright Nanyang Technological University

39

The **main()** function then calls **function1()**

CALL BY REFERENCE – EXAMPLE 2

```
int main() {  
    int x, y;  
    x = 5; y = 5;  
    function1(x, &y);  
    return 0;  
}
```



```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

Content Copyright Nanyang Technological University

40

by passing the value of `x` (i.e. 5)

CALL BY REFERENCE – EXAMPLE 2

```
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
```

Address = 2002 x
5
y
Address = 2048 5

```
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
```

Content Copyright Nanyang Technological University

41

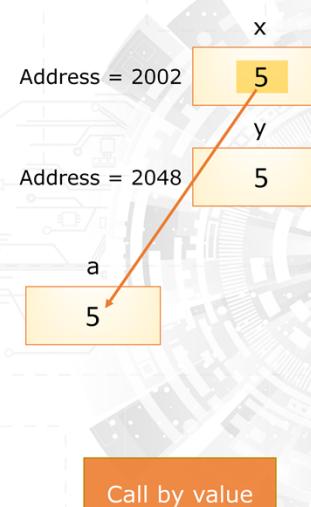
To parameters **a**

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
```

```
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
```



Content Copyright Nanyang Technological University

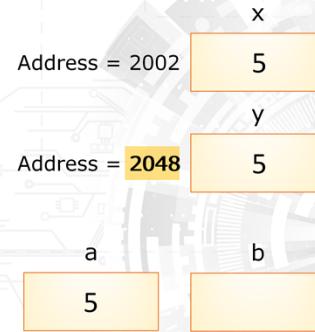
42

The mode of parameter passing for **a** is call by value

CALL BY REFERENCE – EXAMPLE 2

```
int main() {  
    int x, y;  
    x = 5; y = 5;  
    function1(x, &y);  
    return 0;  
}
```

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```



Content Copyright Nanyang Technological University

43

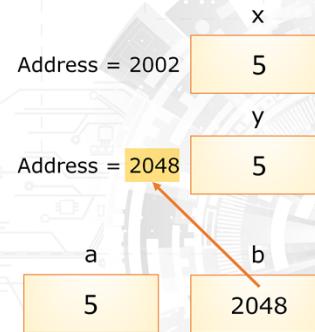
The address of **y** (i.e. 2048) is passed to parameter **b**.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
int main() {  
    int x, y;  
    x = 5; y = 5;  
    function1(x, &y);  
    return 0;  
}
```

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```



Call by reference:
Address, not actual value

Content Copyright Nanyang Technological University

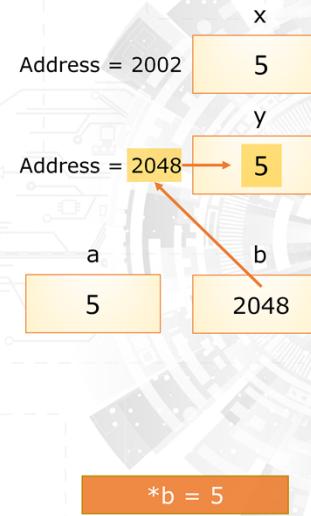
44

The mode of parameter passing for **b** is call by reference. As such, the parameter **b** refers to the memory location of **y** in the **main()** function.

CALL BY REFERENCE – EXAMPLE 2

```
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
```

```
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
```



Content Copyright Nanyang Technological University

45

The value of **b* is 5.

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

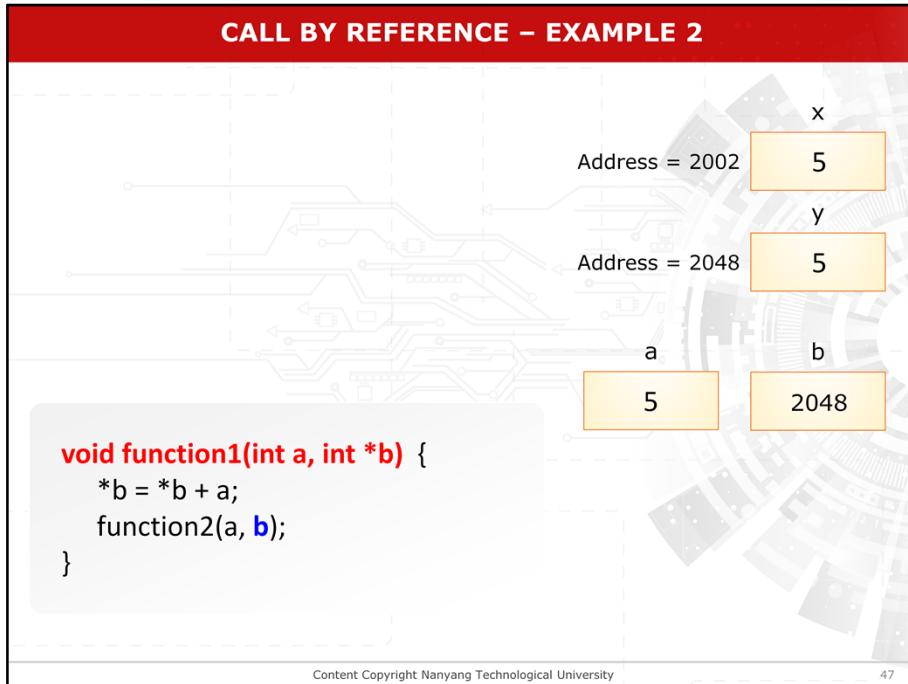
	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-

Content Copyright Nanyang Technological University

46

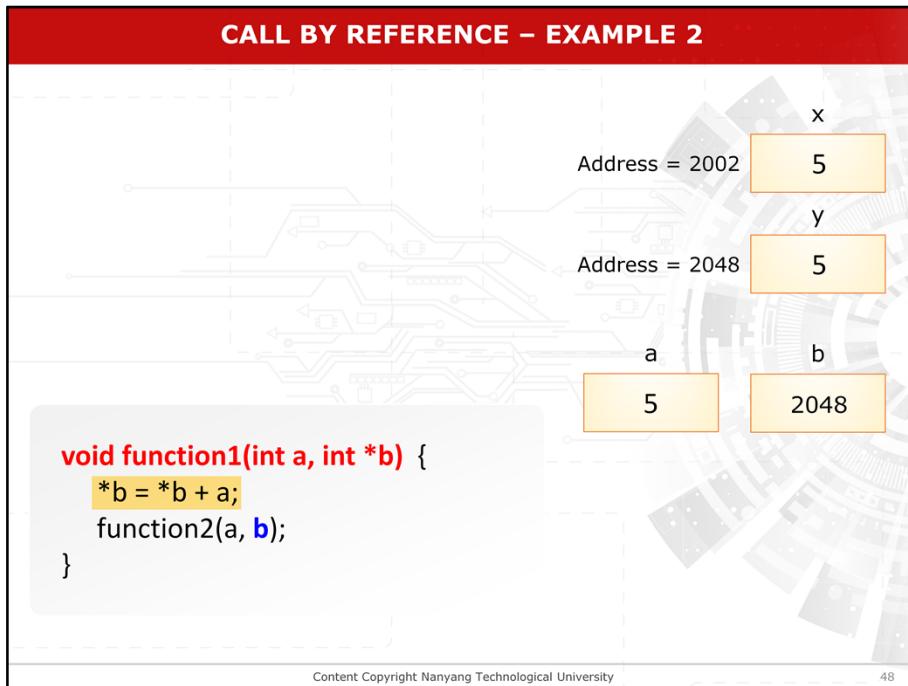
When function1 is called, values of x, y, a and *b are all 5.

CALL BY REFERENCE – EXAMPLE 2



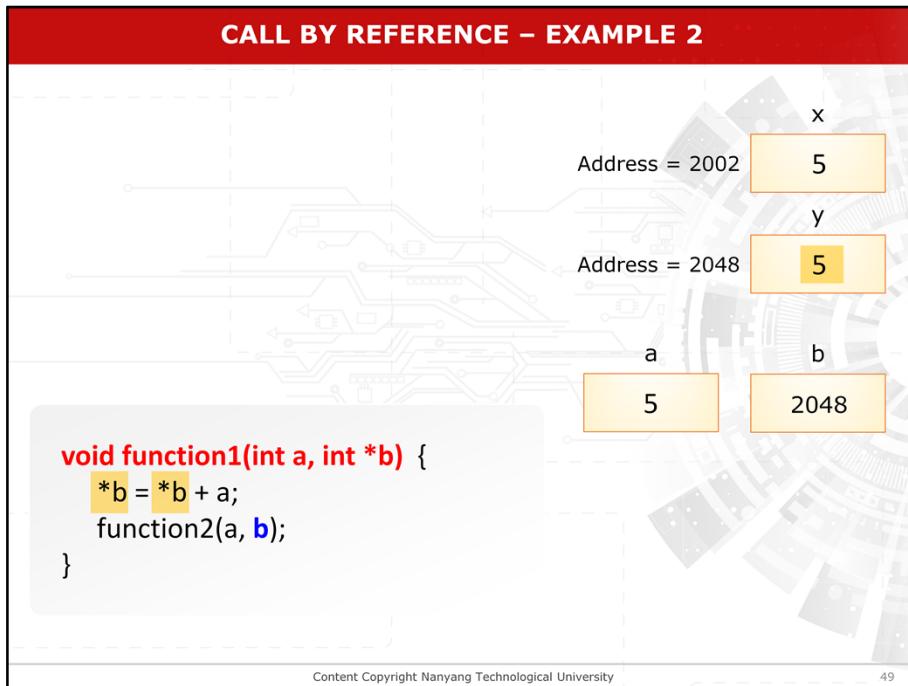
When `function1()` is executed,

CALL BY REFERENCE – EXAMPLE 2



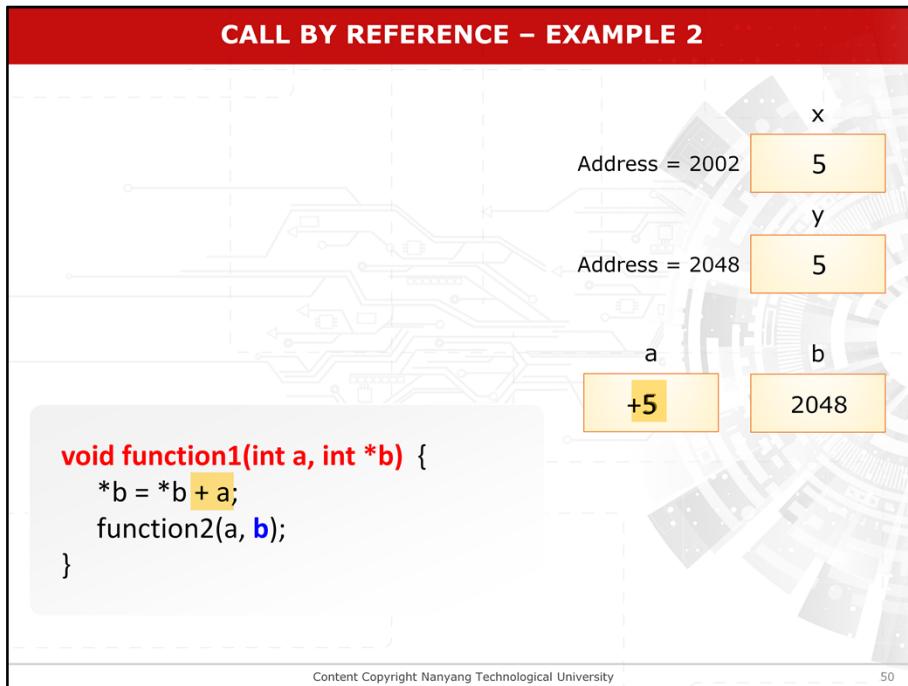
the statement `*b = *b + a` will update the value of

CALL BY REFERENCE – EXAMPLE 2



$*b = 5$

CALL BY REFERENCE – EXAMPLE 2

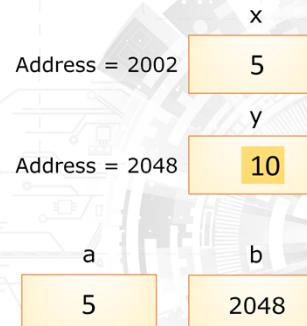


+ 5

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

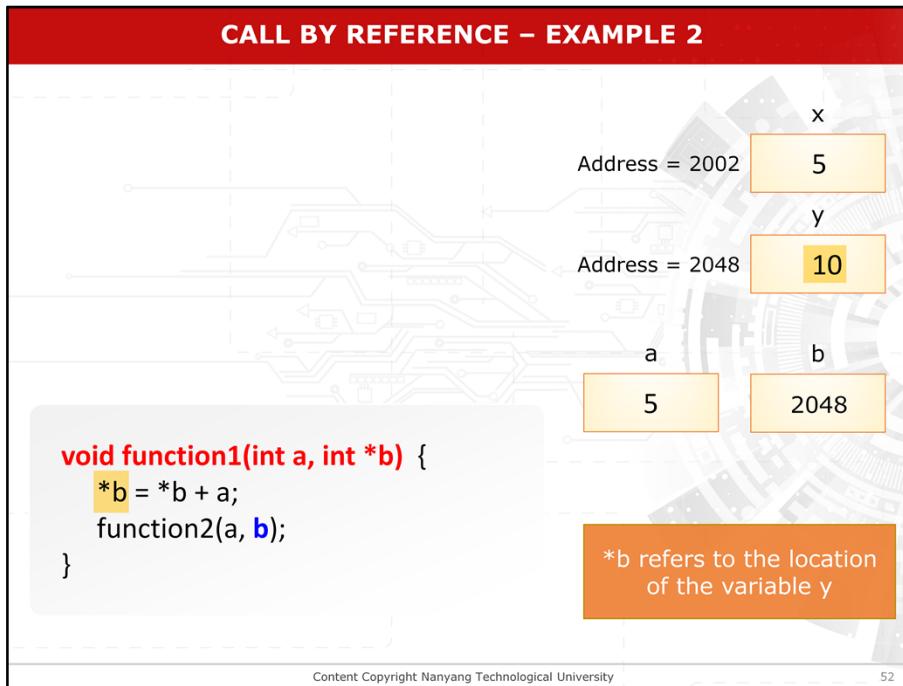


Content Copyright Nanyang Technological University

51

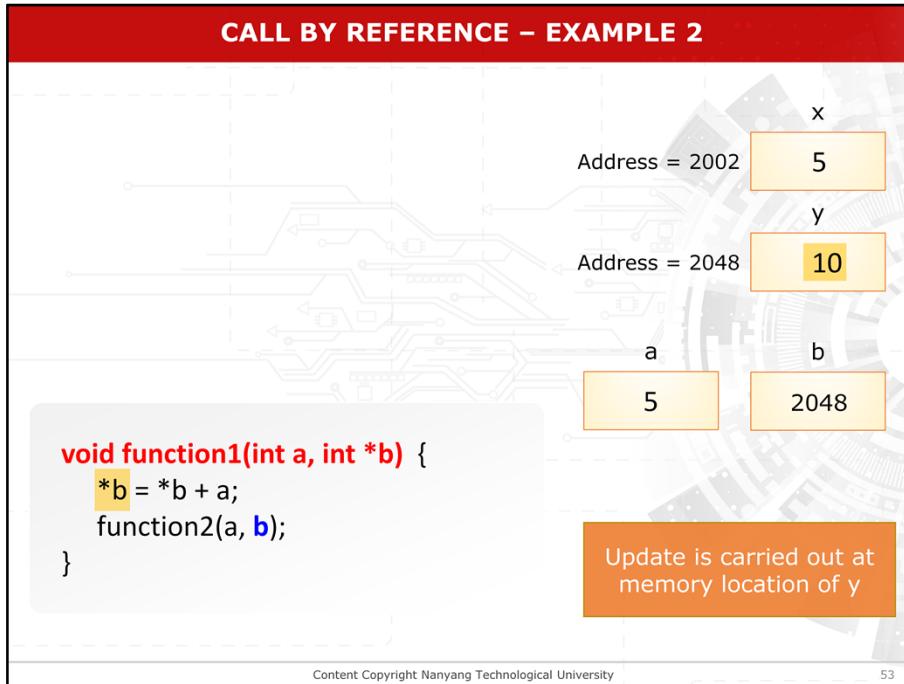
= 10

CALL BY REFERENCE – EXAMPLE 2

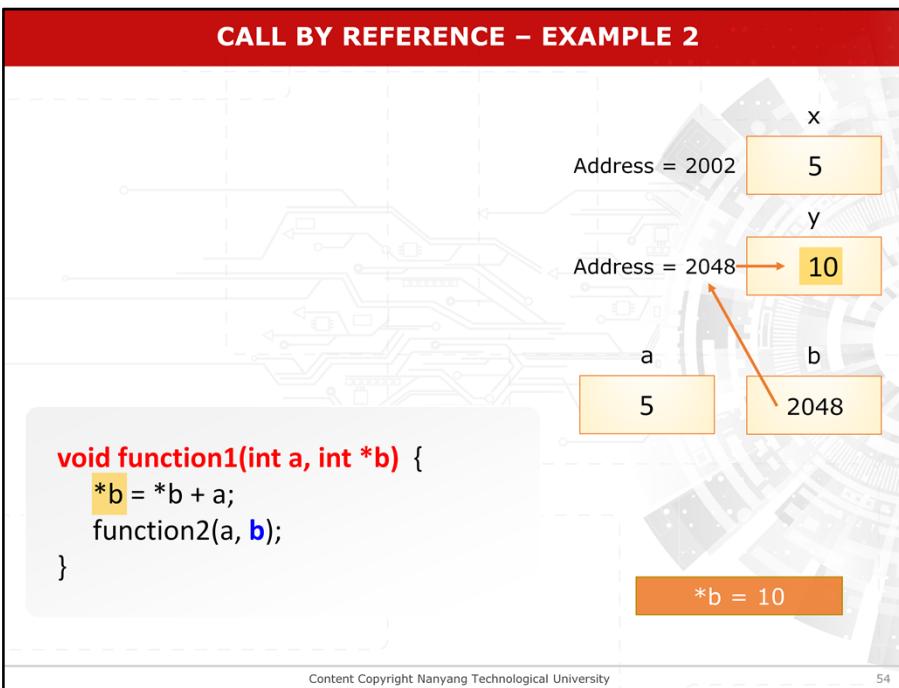


As the pointer variable **b** refers to the location of the variable **y** in the **main()** function

CALL BY REFERENCE – EXAMPLE 2



the update in fact is carried out at the memory location of **y**.



Therefore, the value of **y** = 10 in **main()**, and the value of ***b** = 10 in **function1()**. There is no change in the value of the variable **a** which is 5.

CALL BY REFERENCE – EXAMPLE 2

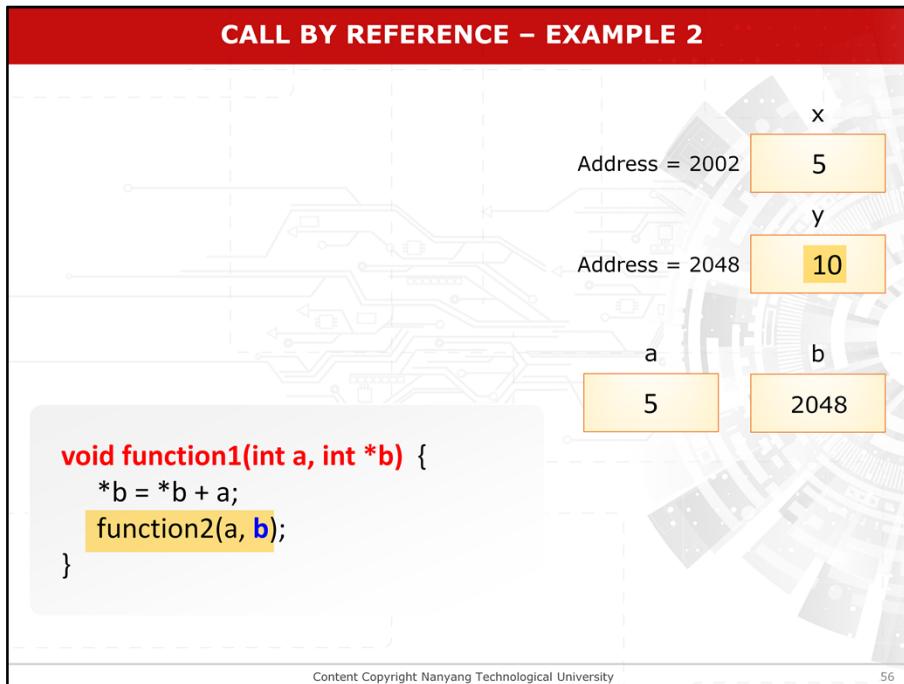
Tracking of the changes in the parameters

Content Copyright Nanyang Technological University

55

So by the time function1 is executed, values of x and a remain as 5 while y and *b become 10.

CALL BY REFERENCE – EXAMPLE 2



After that, **function1()** calls **function2()**

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

x	5
Address = 2002	
y	10
Address = 2048	

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

a	5
b	2048
c	
d	

Content Copyright Nanyang Technological University

57

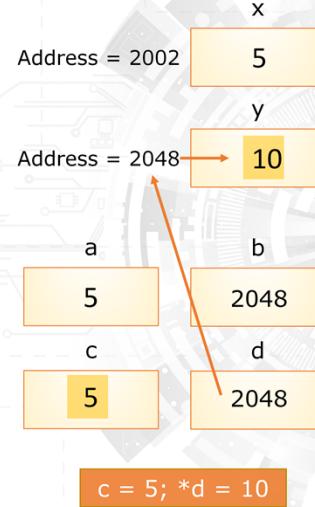
the values of **a** and **b** are passed into the parameters **c** and **d** respectively.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```



Content Copyright Nanyang Technological University

58

The values of *c* is 5 and **d* is 10

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-

Content Copyright Nanyang Technological University

59

The tracking table is updated as shown.

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

x	5
Address = 2002	
y	10
Address = 2048	

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

a	5
c	5
b	2048
d	2048

Content Copyright Nanyang Technological University

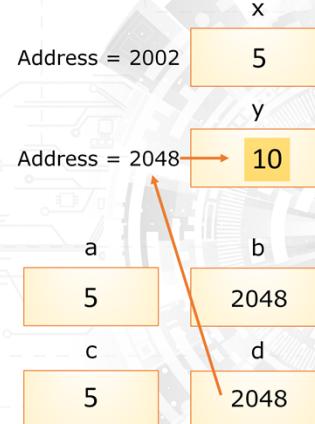
60

When **function2()** is executed, the statement ***d = *d * c;** will update the value of

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```



Content Copyright Nanyang Technological University

61

$*d = 10$

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

x	5
y	10
Address = 2002	
a	5
b	2048
c	* 5
d	2048

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

Content Copyright Nanyang Technological University

62

Multiply by 5

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
void function1(int a, int *b) {  
    *b = *b + a;  
    function2(a, b);  
}
```

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

x	5
y	50
a	5
b	2048
c	5
d	2048

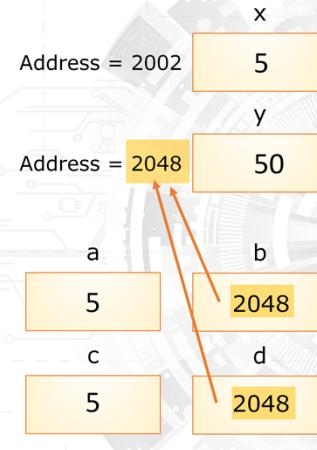
Content Copyright Nanyang Technological University

63

= 50

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```



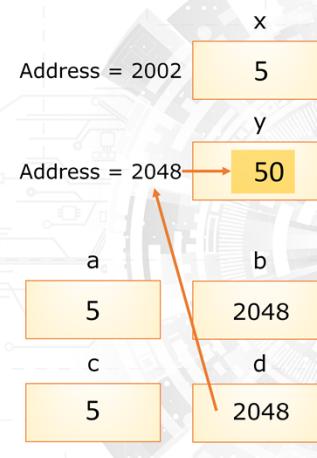
Content Copyright Nanyang Technological University

64

As the pointer variable **d** contains the same address value as **b** in **function1()**, it also refers to the location of the variable **y** in the **main()** function.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```



Content Copyright Nanyang Technological University

65

The update in fact is carried out at the memory location of *y*. Therefore, the value of *y* is also 50 (in **main()**), and the value of **d* = 50 (in **function2()**). There is no change in the value of the variable *c* which is 5.

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-
(v)	5	50	5	50	5	50	-	-

Content Copyright Nanyang Technological University

66

Let's update the tracking table as shown.

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```

x	5
y	50
a	5
b	2048
c	5
d	2048
h	5
k	2048

Content Copyright Nanyang Technological University

67

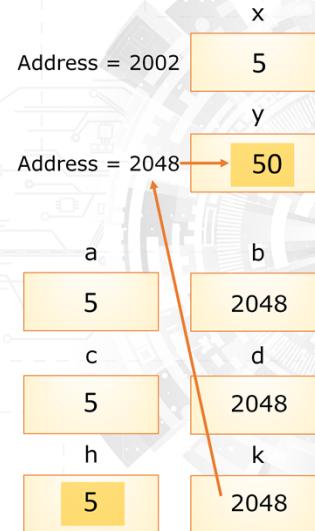
After that, **function2()** calls **function3()** by passing in the values of **c** and **d** to parameters **h** and **k**.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```



Content Copyright Nanyang Technological University

68

The value of **h** is 5 and the value of ***k** is 50

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-
(v)	5	50	5	50	5	50	-	-
(vi)	5	50	5	50	5	50	5	50

Content Copyright Nanyang Technological University

69

The tracking table is updated as shown.

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```

x	5
y	50
a	5
b	2048
c	5
d	2048
h	5
k	2048

Content Copyright Nanyang Technological University

70

When **function3()** is executed,

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```

x	5
y	50
a	5
b	2048
c	5
d	2048
h	5
k	2048

Content Copyright Nanyang Technological University

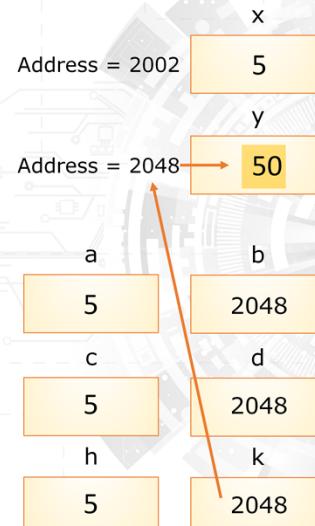
71

the statement `*k = *k - h;` will update the

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```



Content Copyright Nanyang Technological University

72

value of ***k** = 50

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```

x	5
y	50
a	5
b	2048
c	5
d	2048
h	-5
k	2048

Content Copyright Nanyang Technological University

73

-5

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
void function2(int c, int *d) {  
    *d = *d * c;  
    function3(c, d);  
}
```

```
void function3(int h, int *k) {  
    *k = *k - h;  
}
```

x	5
y	45
a	5
b	2048
c	5
d	2048
h	5
k	2048

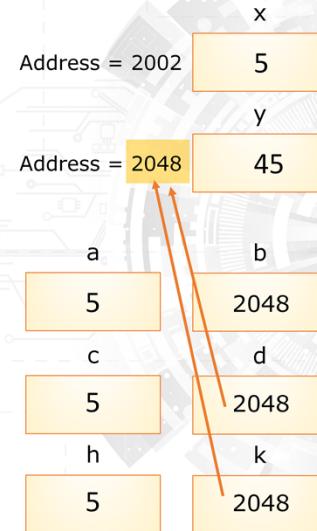
Content Copyright Nanyang Technological University

74

=45.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```



Content Copyright Nanyang Technological University

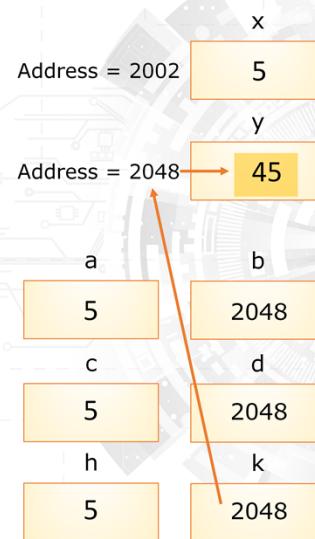
75

As the pointer variable **k** contains the same address value as **d** in **function2()**, it also refers to the location of the variable **y** in the **main()** function.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

*k = 45



Content Copyright Nanyang Technological University

76

The update in fact is carried out at the memory location of *y*. Therefore, the value of *y* is also 45 (in **main()**), and the value of **k* = 45 (in **function3()**). There is no change in the value of the variable *h* which is 5.

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-
(v)	5	50	5	50	5	50	-	-
(vi)	5	50	5	50	5	50	5	50
(vii)	5	45	5	45	5	45	5	45

Content Copyright Nanyang Technological University

77

The tracking table is updated as shown.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

x	5
Address = 2002	
y	45
Address = 2048	
a	b
5	2048
c	d
5	2048
h	k
5	2048

Content Copyright Nanyang Technological University

78

After that, **function3()** finishes the execution and terminates.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

x	5
Address = 2002	
y	45
Address = 2048	
a	b
5	2048
c	d
5	2048
h	k
5	2048

Content Copyright Nanyang Technological University

79

Parameters h and k are released.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-
(v)	5	50	5	50	5	50	-	-
(vi)	5	50	5	50	5	50	5	50
(vii)	5	45	5	45	5	45	5	45
(viii)	5	45	5	45	5	45	-	-

Content Copyright Nanyang Technological University

80

The tracking table is updated as shown.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

x	5
Address = 2002	
y	45
Address = 2048	
a	5
b	2048
c	5
d	2048

Content Copyright Nanyang Technological University

81

The control passes back to **function2()** for execution and then terminates.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

x	5
Address = 2002	
y	45
Address = 2048	
a	5
b	2048
c	5
d	2048

Content Copyright Nanyang Technological University

82

Parameters c and d are released

[+animations]

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-
(v)	5	50	5	50	5	50	-	-
(vi)	5	50	5	50	5	50	5	50
(vii)	5	45	5	45	5	45	5	45
(viii)	5	45	5	45	5	45	-	-
(ix)	5	45	5	45	-	-	-	-

Content Copyright Nanyang Technological University

83

The tracking table is updated as shown.

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```

x	5
Address = 2002	
y	45
Address = 2048	
a	5
b	2048

Content Copyright Nanyang Technological University

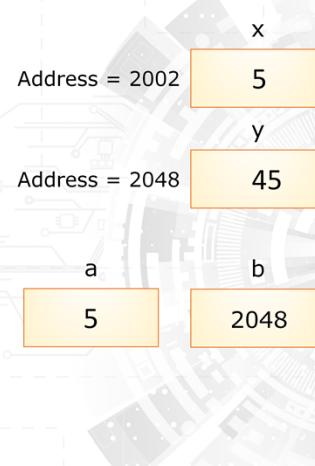
84

The control passes back to **function1()**, and then terminates and returns to the **main()** function.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

```
#include <stdio.h>
void function1 (int, int); void function2 (int, int);
void function3 (int, int);
int main() {
    int x, y;
    x = 5; y = 5;
    function1(x, &y);
    return 0;
}
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
void function2(int c, int *d) {
    *d = *d * c;
    function3(c, d);
}
void function3(int h, int *k) {
    *k = *k - h;
}
```



Content Copyright Nanyang Technological University

85

Parameter a and b are released.

[+animations]

CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

	x	y	a	*b	c	*d	h	*k
(i)	5	5	-	-	-	-	-	-
(ii)	5	5	5	5	-	-	-	-
(iii)	5	10	5	10	-	-	-	-
(iv)	5	10	5	10	5	10	-	-
(v)	5	50	5	50	5	50	-	-
(vi)	5	50	5	50	5	50	5	50
(vii)	5	45	5	45	5	45	5	45
(viii)	5	45	5	45	5	45	-	-
(ix)	5	45	5	45	-	-	-	-
(x)	5	45	-	-	-	-	-	-

Content Copyright Nanyang Technological University

86

The tracking table is updated as shown. So from this example, we see the effect of using call by reference.

CALL BY REFERENCE – WHEN TO USE?

When do we use call by reference?

Content Copyright Nanyang Technological University

87

When do we use call by reference?

CALL BY REFERENCE – WHEN TO USE?

Use call by reference:

- When you need to pass **more than one value** back from a function

Generally, call by reference is used when we need to pass more than one value back from a function,

CALL BY REFERENCE – WHEN TO USE?

Use call by reference:

- When you need to pass **more than one value** back from a function
- When using call by value will result in a **large piece of information** being **copied** to the formal parameter, for efficiency reason, for example, passing large arrays

Content Copyright Nanyang Technological University

89

or in the case that when we use call by value, it will result in a large piece of information being copied to the parameter. This could happen when we pass a large array size or structure record.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Content Copyright Nanyang Technological University

90

We have seen examples on using indirection operator. Double indirection is also quite commonly used in C programming. We will use this example to illustrate the use of double indirection.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Address = 1024

2

Content Copyright Nanyang Technological University

91

Parameter a has been declared to be of integer value 2.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Address = 1024

a
2

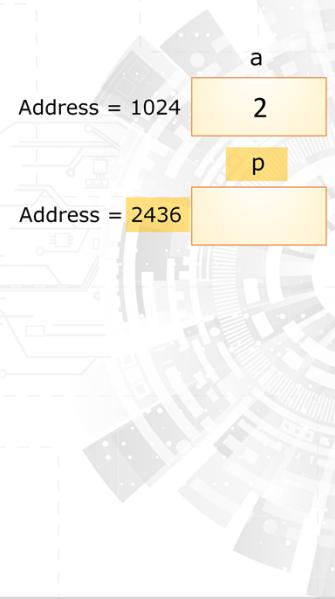
Content Copyright Nanyang Technological University

92

Let's assume the memory address for parameter a is 1024.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```



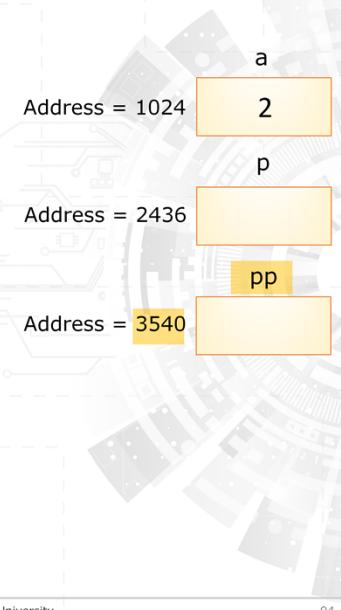
Content Copyright Nanyang Technological University

93

Next, let's assume parameter p was declared as a pointer variable at memory address 2436.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```



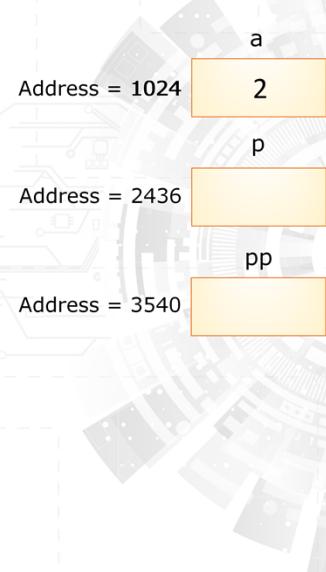
Content Copyright Nanyang Technological University

94

Let's assume parameter p p was declared as a pointer variable at memory address 3540.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```



Content Copyright Nanyang Technological University

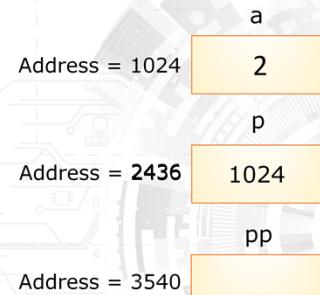
95

In the highlighted statement, the value of the address of a is stored in parameter p.

[+animations]

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```



Content Copyright Nanyang Technological University

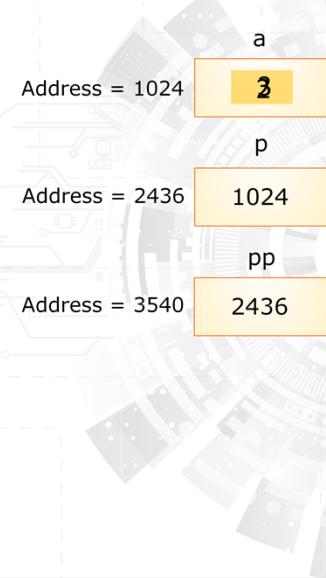
96

In the highlighted statement, the value of the address of p is stored in parameter pp.

[+animations]

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```



Content Copyright Nanyang Technological University

97

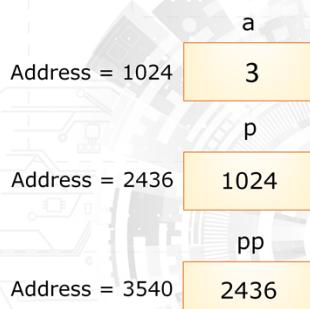
When a is increased by 1, the value of a changes from 2 to 3.

[+animations]

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Output



Content Copyright Nanyang Technological University

98

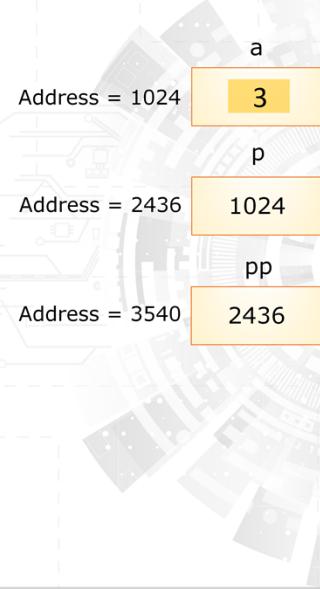
Let's see what are the outputs.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Output

```
a = 3
```



Content Copyright Nanyang Technological University

99

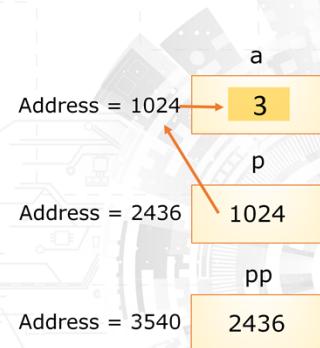
The value of a is 3.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Output

```
a = 3
*p = 3
```



Using pointer dereferencing
*p = value stored in the address pointed to = 3

Content Copyright Nanyang Technological University

100

Using pointer dereferencing, *p will have the value of 3.

DOUBLE INDIRECTION – EXAMPLE

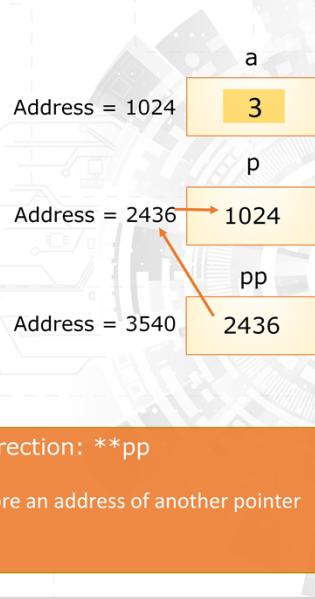
```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Output

```
a = 3
*p = 3
```

Double indirection: **pp

Variable **pp** will be used to store an address of another pointer variable (in this case **p**)



Content Copyright Nanyang Technological University

101

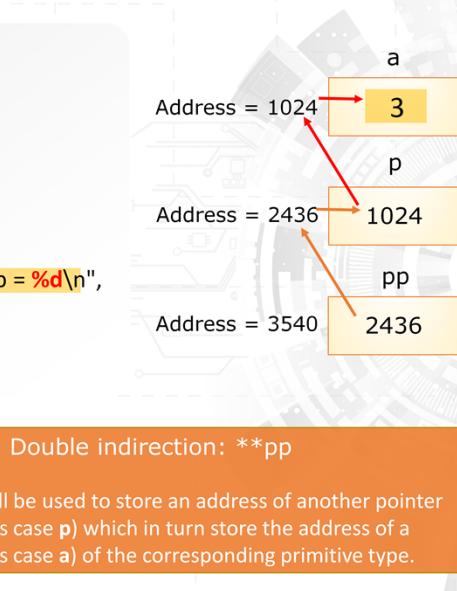
The ****pp** refers to double indirection. The variable **pp** will be used to store an address of another pointer variable (in this case **p**)

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Output

```
a = 3
*p = 3
```



Content Copyright Nanyang Technological University

102

Which in turn store the address of a variable (in this case **a**) of the corresponding primitive type.

DOUBLE INDIRECTION – EXAMPLE

```
#include <stdio.h>
int main()
{
    int a=2, *p, **pp;
    p = &a;
    pp = &p;
    a++;
    printf("a = %d\n, *p = %d\n, **pp = %d\n",
           a, *p, **pp);
    return 0;
}
```

Output

```
a = 3
*p = 3
**pp = 3
```

Double indirection: **pp

Variable **pp** will be used to store an address of another pointer variable (in this case **p**) which in turn store the address of a variable (in this case **a**) of the corresponding primitive type.

Content Copyright Nanyang Technological University

103

Therefore, ****pp** will have the value stored in **a**, that is, 3.

This example shows the use of double indirection.

PROGRAMMING PROBLEM

How to write a function to swap the contents of 2 variables?

Content Copyright Nanyang Technological University

104

A programming problem given is to write a function that can swap the values of two parameters of the function.

PROGRAMMING PROBLEM

```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

a	5
b	10
temp	
Address = 1424	

Content Copyright Nanyang Technological University

105

As shown in this program, variables a and b have been declared and assigned values accordingly. Variable temp has been declared as well.

PROGRAMMING PROBLEM

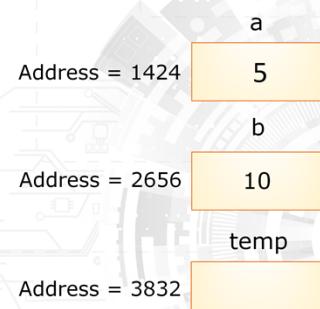
```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

```
Before: a = 5, b = 10
```



Content Copyright Nanyang Technological University

106

Executing the printf statement will give the output as shown.

PROGRAMMING PROBLEM

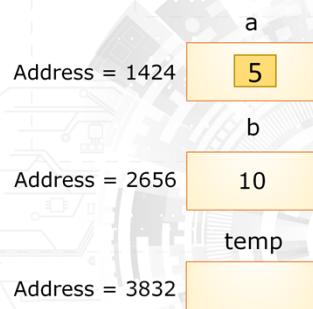
```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

```
Before: a = 5, b = 10
```



Content Copyright Nanyang Technological University

107

Variable temp has been assigned the value of a

[+animations]

PROGRAMMING PROBLEM

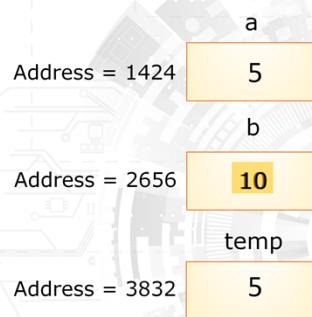
```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

```
Before: a = 5, b = 10
```



Content Copyright Nanyang Technological University

108

Then the value of a is reassigned to be value of b. So the value of a changes from 5 to 10.

[+animations]

PROGRAMMING PROBLEM

```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

```
Before: a = 5, b = 10
```

The diagram illustrates the state of memory after the execution of the program. It shows four memory locations with their addresses and corresponding values:

- Address = 1424: Value 10, labeled 'a'.
- Address = 2656: Value 10, labeled 'b'.
- Address = 3832: Value 5, labeled 'temp'.

Content Copyright Nanyang Technological University

109

Then the value of b is reassigned to be value of temp. So the value of b change from 10 to 5.

[+animations]

PROGRAMMING PROBLEM

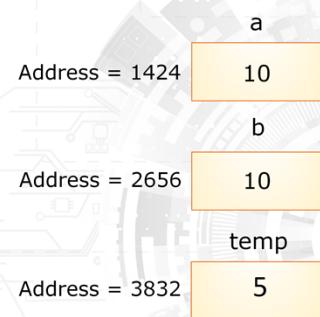
```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

```
Before: a = 5, b = 10
After: a = 10, b = 5
```



Content Copyright Nanyang Technological University

110

Thus, the output shows the swap required.

PROGRAMMING PROBLEM

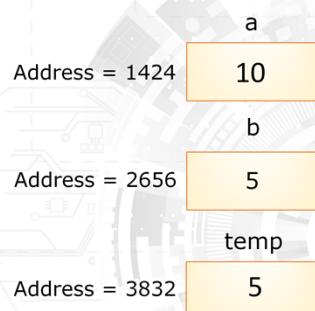
```
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    int temp;

    printf("Before: a = %d, b = %d\n", a, b);
temp = a;    a = b;    b = temp;
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
```

Output

```
Before: a = 5, b = 10
After: a = 10, b = 5
```



Question: How do you implement a function called **swap()** to swap the values of 2 variables?

Content Copyright Nanyang Technological University

111

Question: How do you implement a function called **swap()** to swap the values of 2 variables?

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    /* Try writing your own code */
}
```

Content Copyright Nanyang Technological University

112

In this updated code, a swap function is declared.

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    /* Try writing your own code */
}
```

Address = 1364 a 5
b

Address = 2586 10

Content Copyright Nanyang Technological University 113

In the **main()** function of the program template, it declares two variables **a** and **b**.

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    /* Try writing your own code */
}
```

Address = 1364 5
a
b
Address = 2586 10

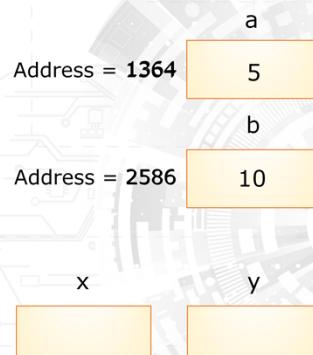
Content Copyright Nanyang Technological University

114

When the function **swap()** is called,

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    /* Try writing your own code */
}
```



Content Copyright Nanyang Technological University

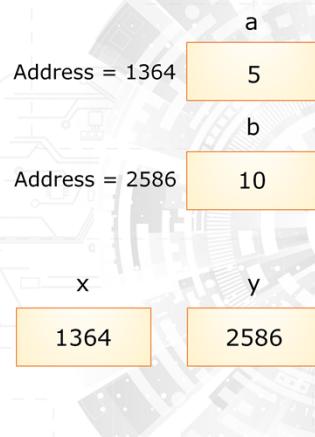
115

the addresses of the two variables are passed into the calling function.

[+animations]

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    /* Try writing your own code */
}
```



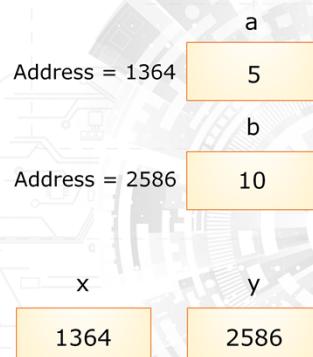
Content Copyright Nanyang Technological University

116

In the function **swap()**, it swaps the contents of the two parameters **x** and **y**, which contain the addresses of the variables **a** and **b** passed in from the **main()** function. It is done via **call by reference**, as the function is required to return two values back to the calling function.

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    /* Try writing your own code */
}
```



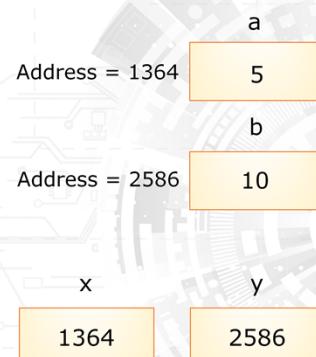
Content Copyright Nanyang Technological University

117

Question: How do you implement the function **swap()**?

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}
```



Content Copyright Nanyang Technological University

118

In the **swap()** function, a simple swap operation is performed based on the parameters **x** and **y** via call by reference.

PROGRAMMING PROBLEM

```

#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}

```

Content Copyright Nanyang Technological University

119

A parameter temp is declared.

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}
```

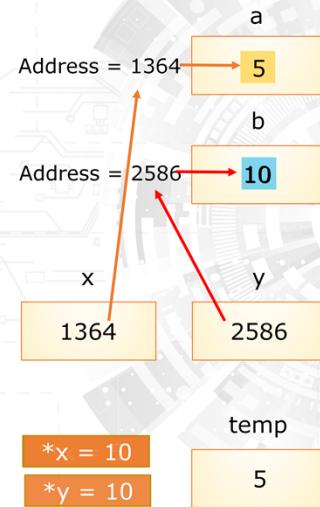
Content Copyright Nanyang Technological University 120

The value of temp contains the value stored in parameter a.

[+animations]

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}
```



Content Copyright Nanyang Technological University

121

The value of *a* changes from 5 to 10.

[+animations]

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}
```

Address = 1364 a
10
b
Address = 2586 → 10
x 1364
y 2586
temp
*y = 5 5

Content Copyright Nanyang Technological University

122

The value of b changes from 10 to 5.

[+animations]

PROGRAMMING PROBLEM

The diagram illustrates the state of memory before and after a swap operation. It shows two pairs of variables: 'a' and 'b', and 'x' and 'y'. Variable 'a' has address 1364 and value 10. Variable 'b' has address 2586 and value 5. Pointers 'x' and 'y' also have addresses 1364 and 2586 respectively. In the swap function, a temporary variable 'temp' is used to store the value of *x (which is 10). Then, *x is assigned the value of *y (which is 5), and *y is assigned the value of 'temp' (which is 10).

```

#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}

```

Content Copyright Nanyang Technological University 123

The temporary variable **temp** is used to store the temporary data during the swapping operation.

PROGRAMMING PROBLEM

```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}
```

Address = 1364 10
a
b
Address = 2586 5
x y
1364 2586
temp
5

Content Copyright Nanyang Technological University

124

This example shows us how to use pointer dereferencing for the swapping operation.

SUMMARY

We have completed the lesson on Call by Reference and you have learnt to:

- Explain the two modes of parameter passing

Content Copyright Nanyang Technological University

125

We have completed the lesson on Call by Reference and you have learnt to:

Explain the two modes of parameter passing

SUMMARY

We have completed the lesson on Call by Reference and you have learnt to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable

Explain that the parameter called by reference is a pointer variable

SUMMARY

We have completed the lesson on Call by Reference and you have learnt to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable
- Apply call by reference in programming

Apply call by reference in programming

SUMMARY

We have completed the lesson on Call by Reference and you have learnt to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable
- Apply call by reference in programming
- Explain double indirection

Explain double indirection

SUMMARY

We have completed the lesson on Call by Reference and you have learnt to:

- Explain the two modes of parameter passing
- Explain that the parameter called by reference is a pointer variable
- Apply call by reference in programming
- Explain double indirection
- Apply pointer dereferencing for swapping operation

And apply pointer dereferencing for swapping operation.