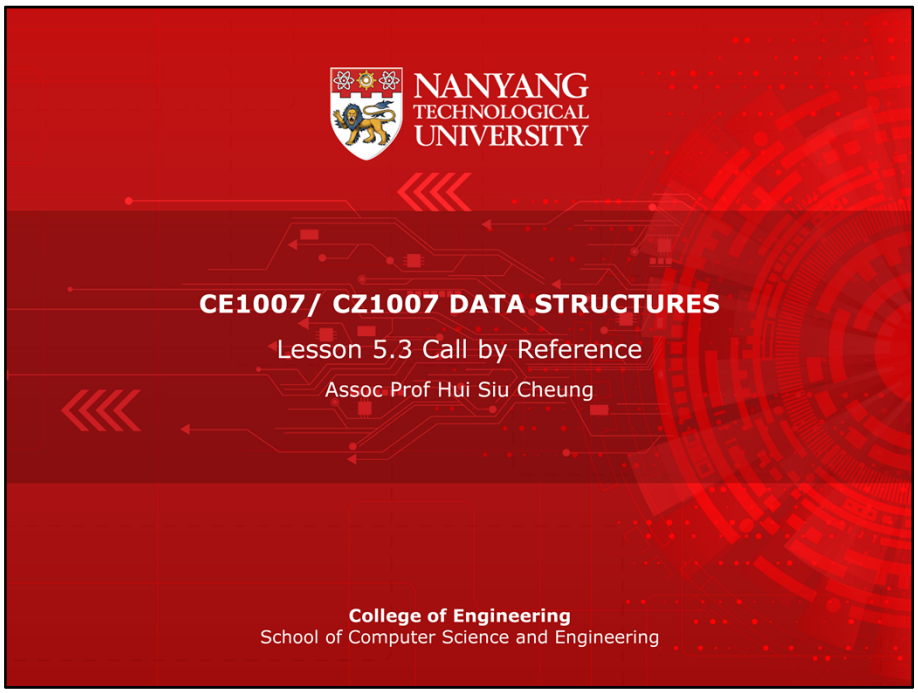
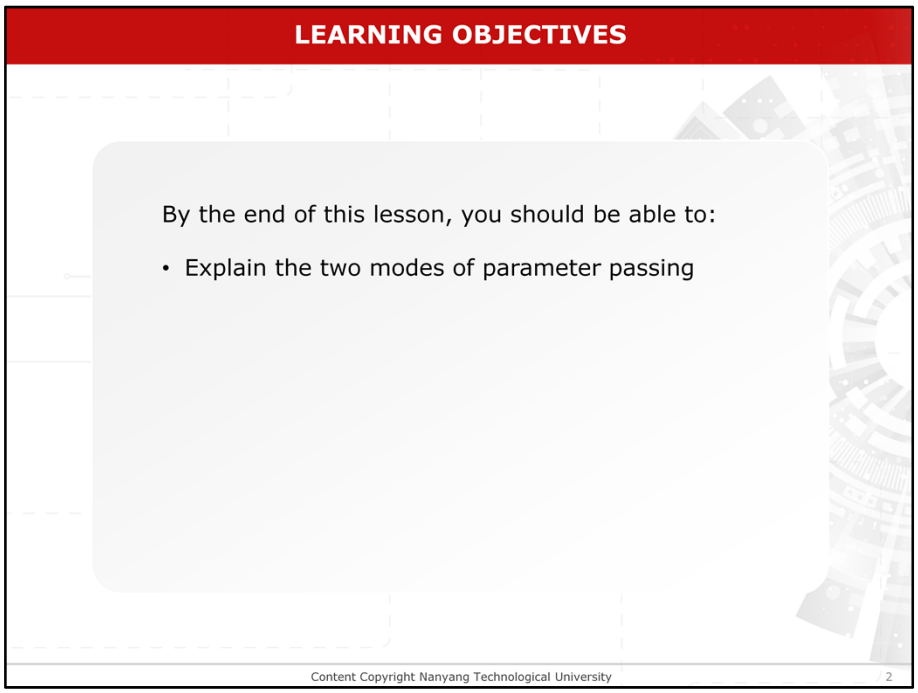
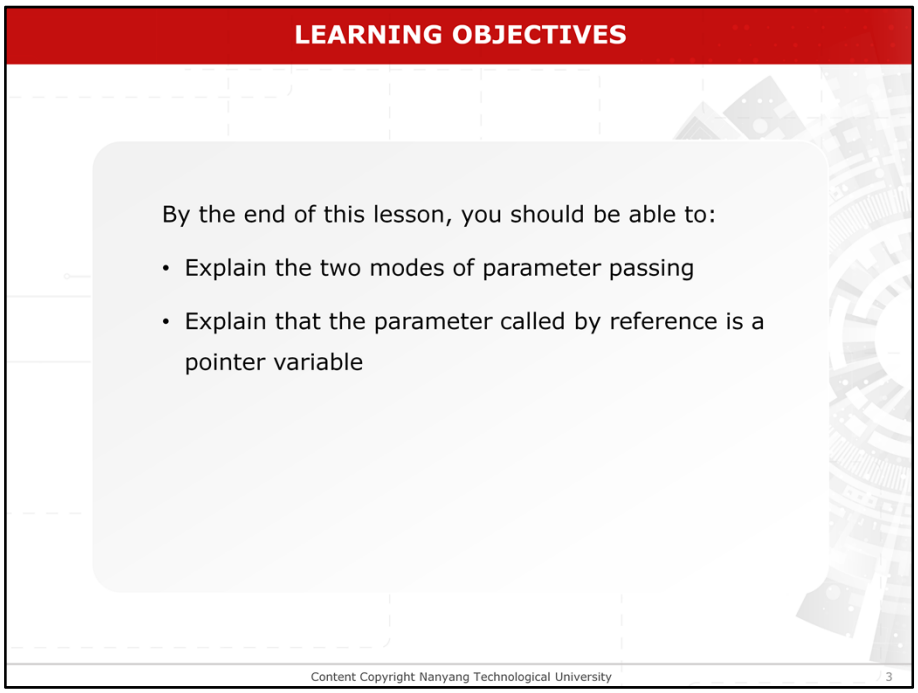

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 5.3 Call by Reference
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering

Lesson 5.3:

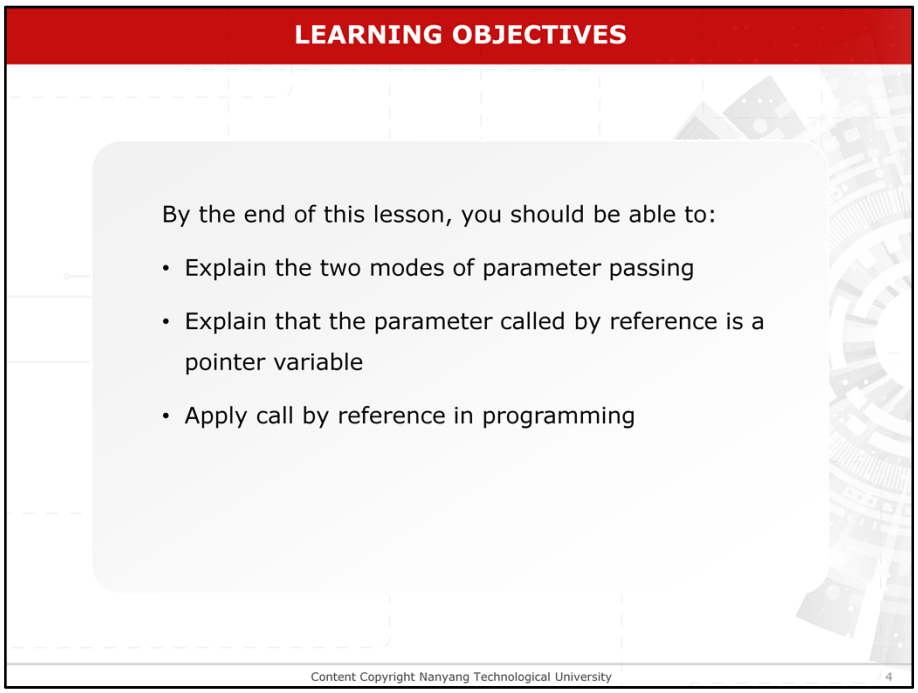



After this lesson, you should be able to:

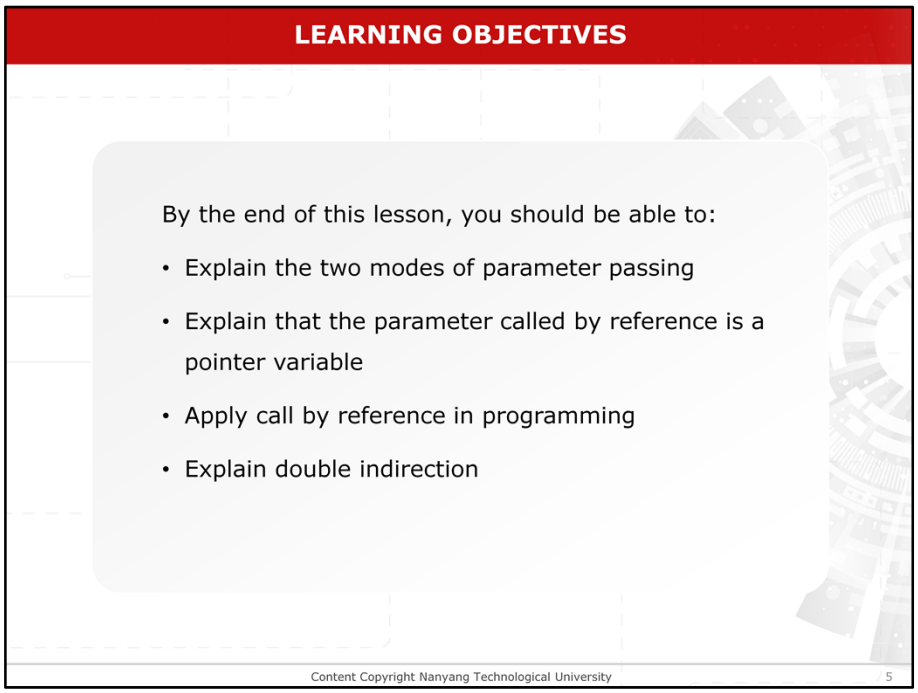
Explain the two modes of parameter passing



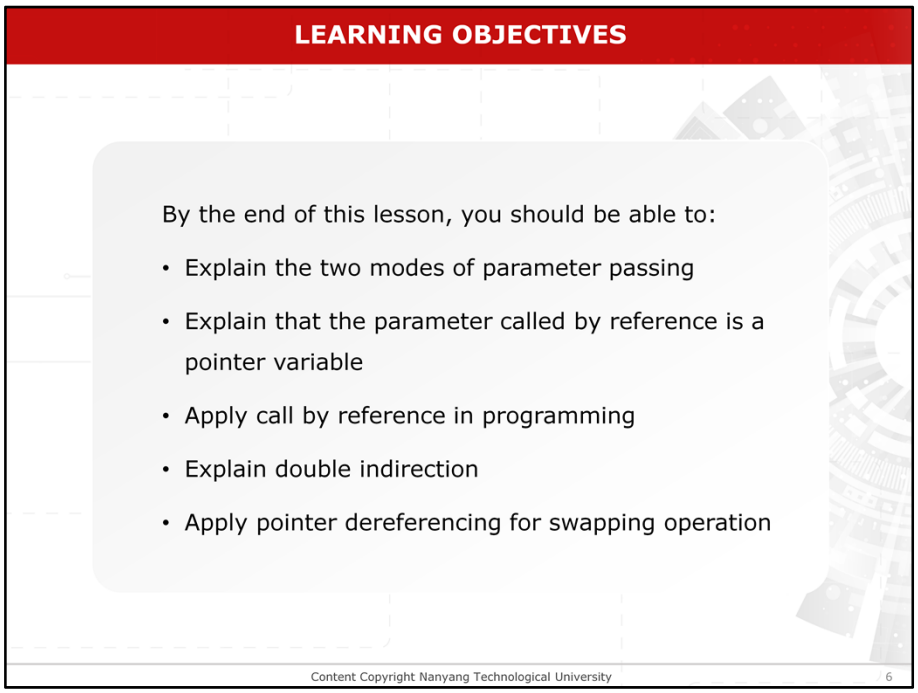

Explain that the parameter called by reference is a pointer variable



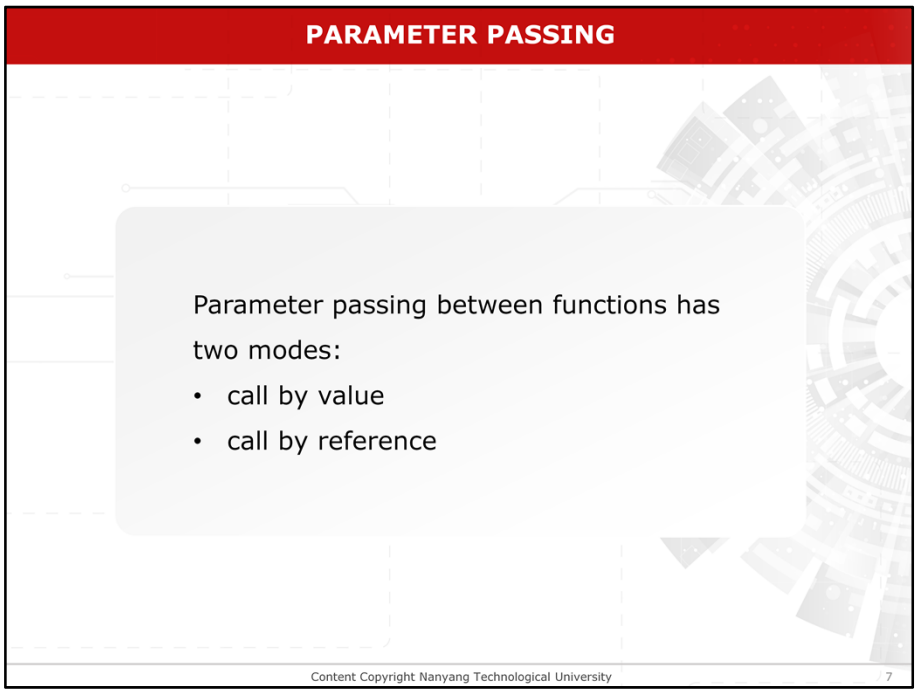

Apply call by reference in programming



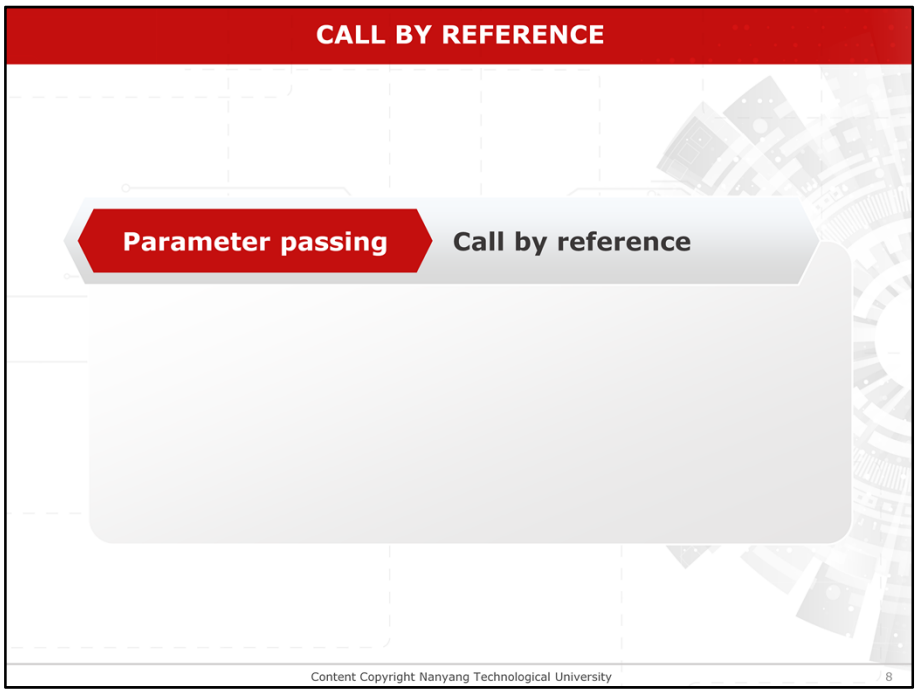

Explain double indirection



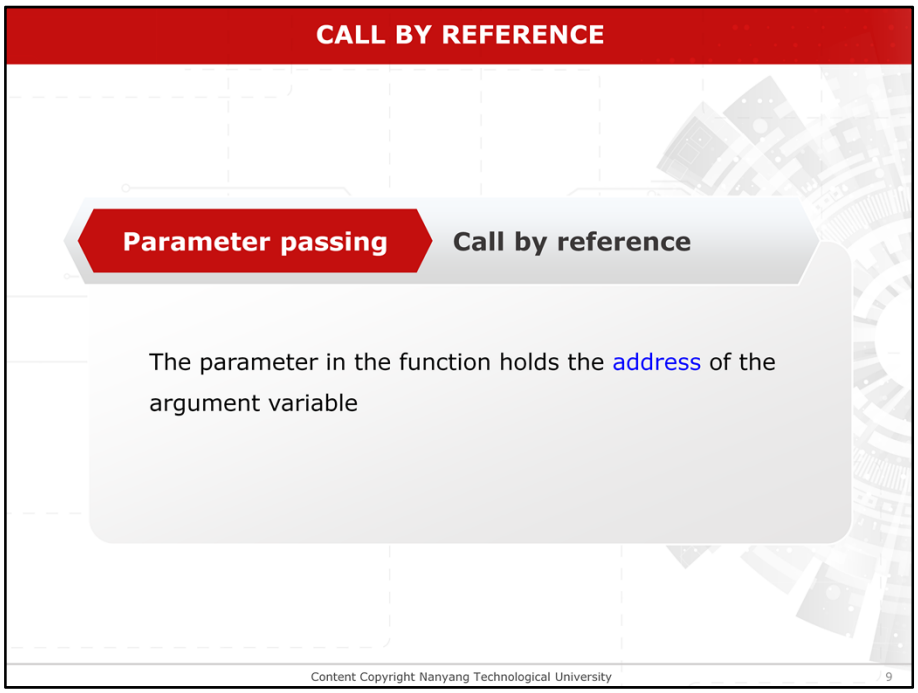

Apply pointer dereferencing for swapping operation



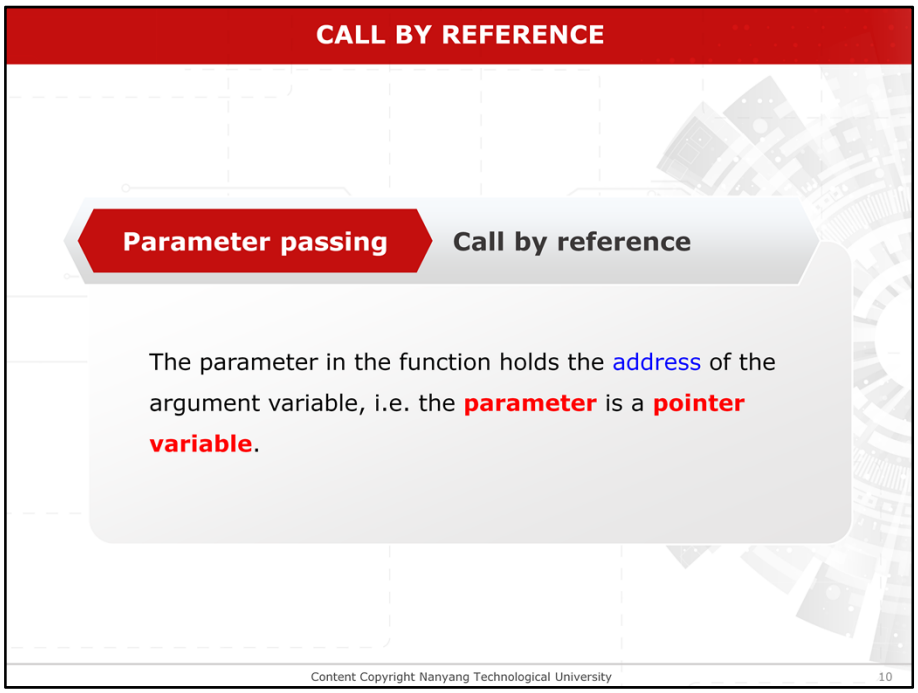

Parameter passing between functions can be done either through call by value or call by reference. Call by value has been discussed in the lesson on functions. This lesson will cover on call by reference.




In call by reference,

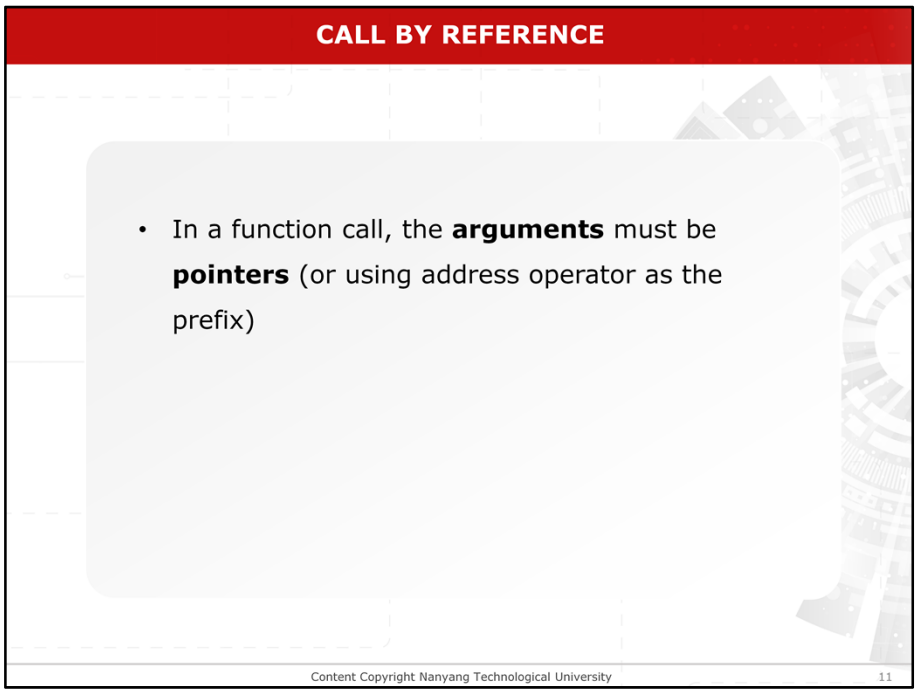



The parameter in the function holds the addresses of the argument variables

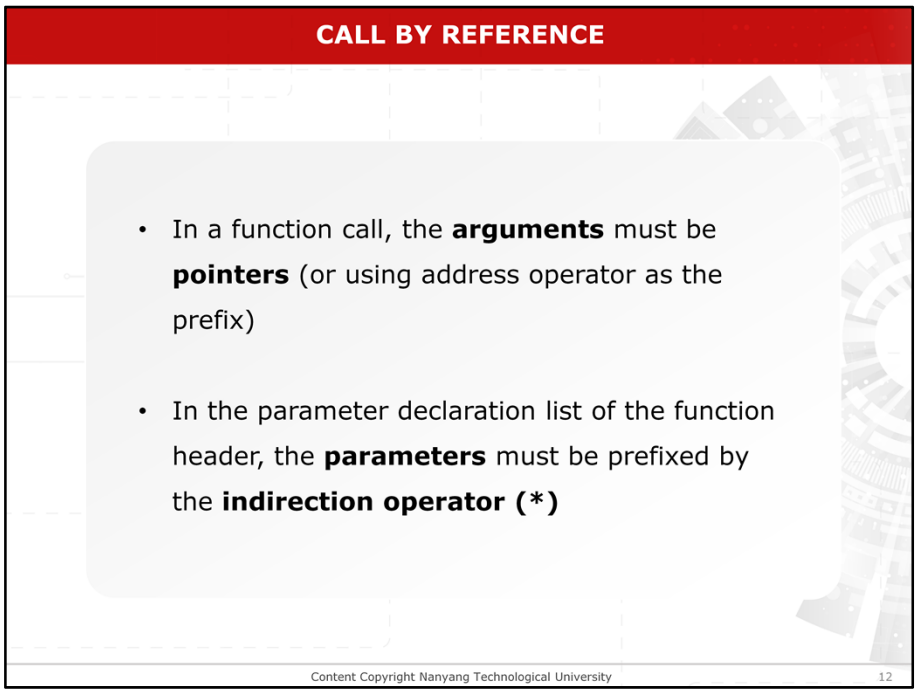



that is, the parameter is a **pointer variable**.

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.




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



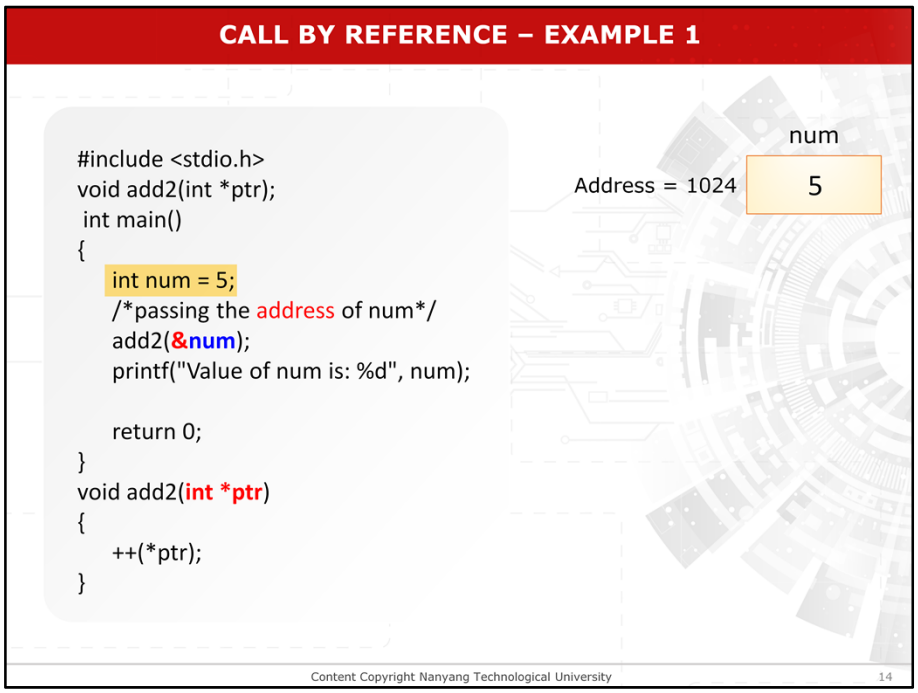

In the parameter declaration list of the function header, the **parameters** must be prefixed by the **indirection operator**.

## 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);
}
```

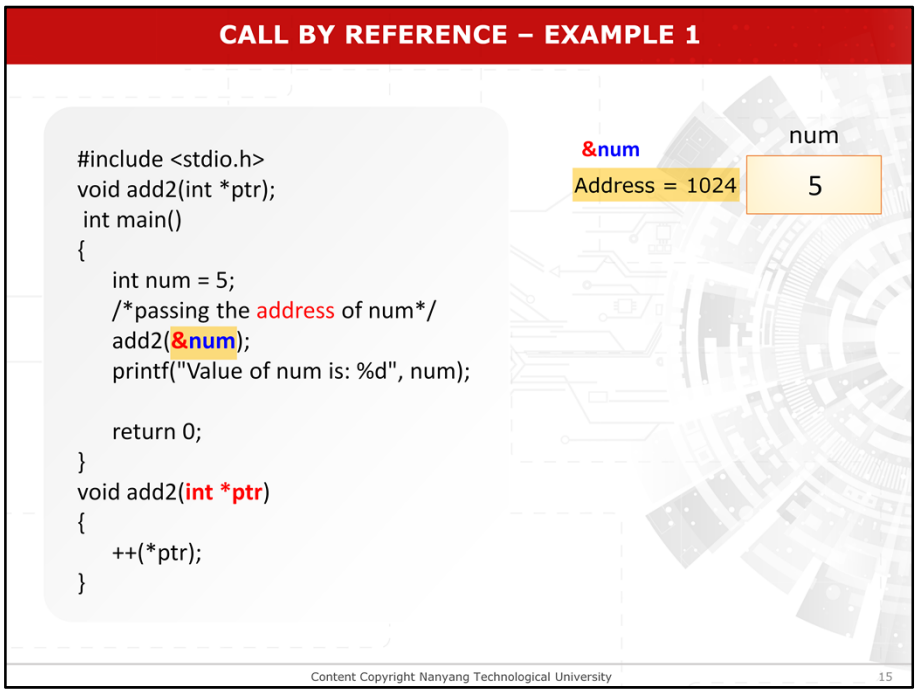



Let us look at this program that uses call by reference and go through it step by step.

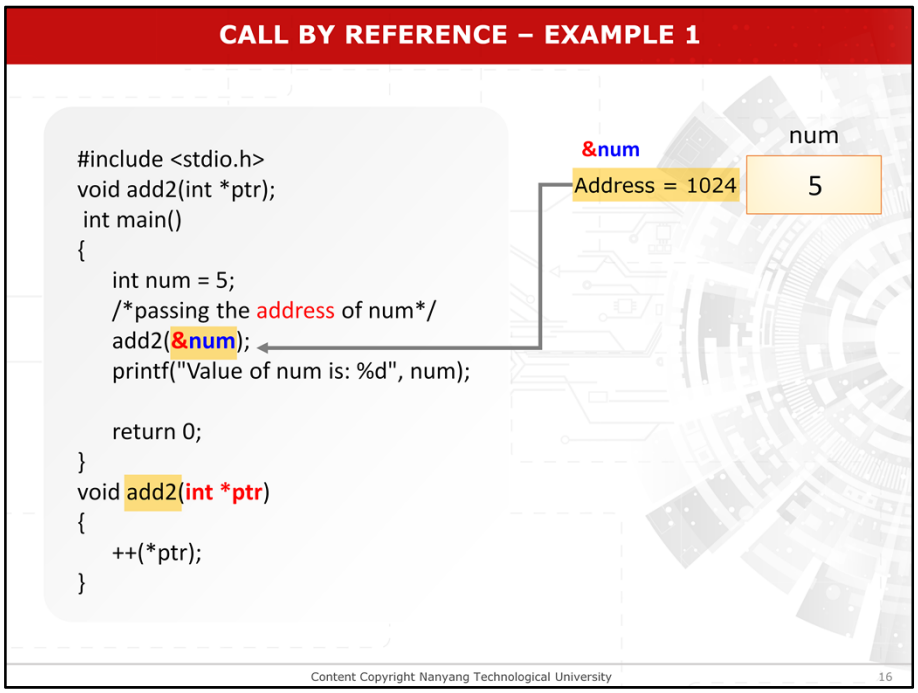



Step 1: the variable **num** is initially assigned with a value 5 in **main()**.

*[+animation]*

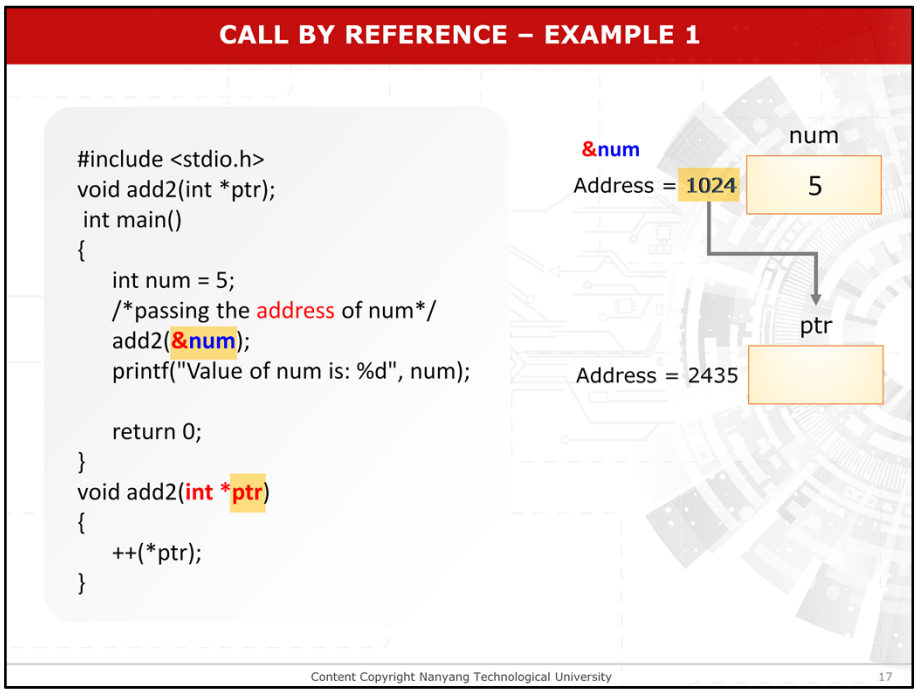



Step 2: The address of the variable **num** is then



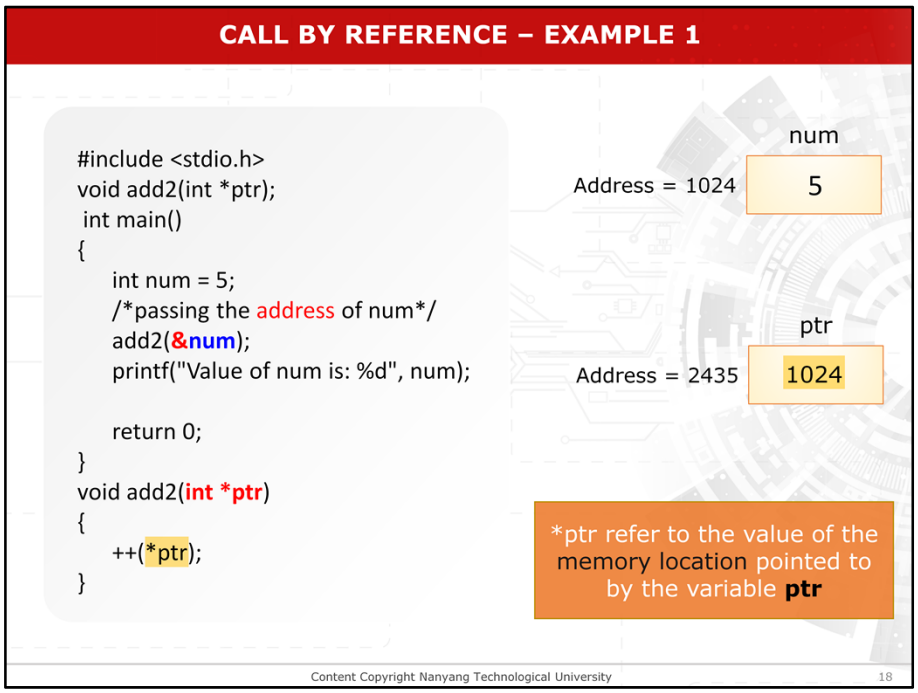

passed as an argument to the function **add2()**

*[+animation]*

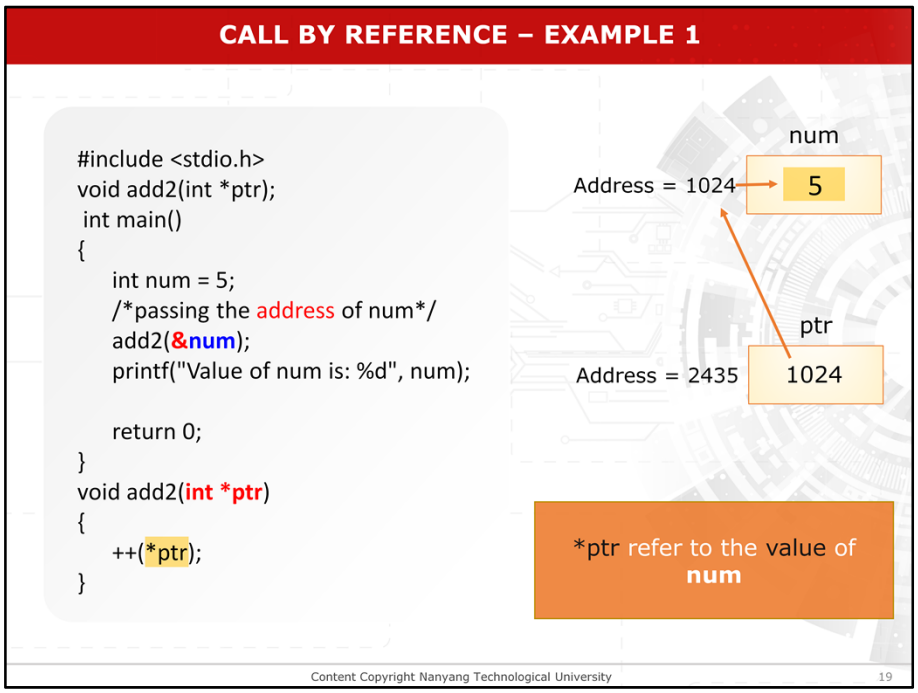



and stored in the parameter **ptr** in the function in step 3.

*[+animation]*




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

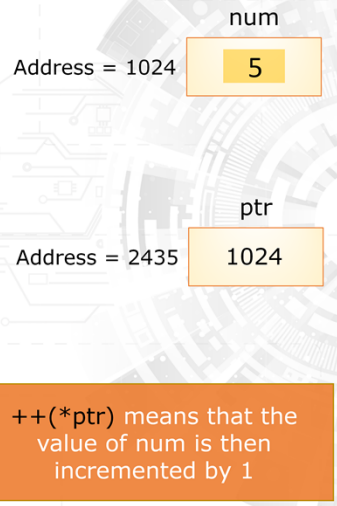



That it **num**

*[+animation]*

## CALL BY REFERENCE – EXAMPLE 1

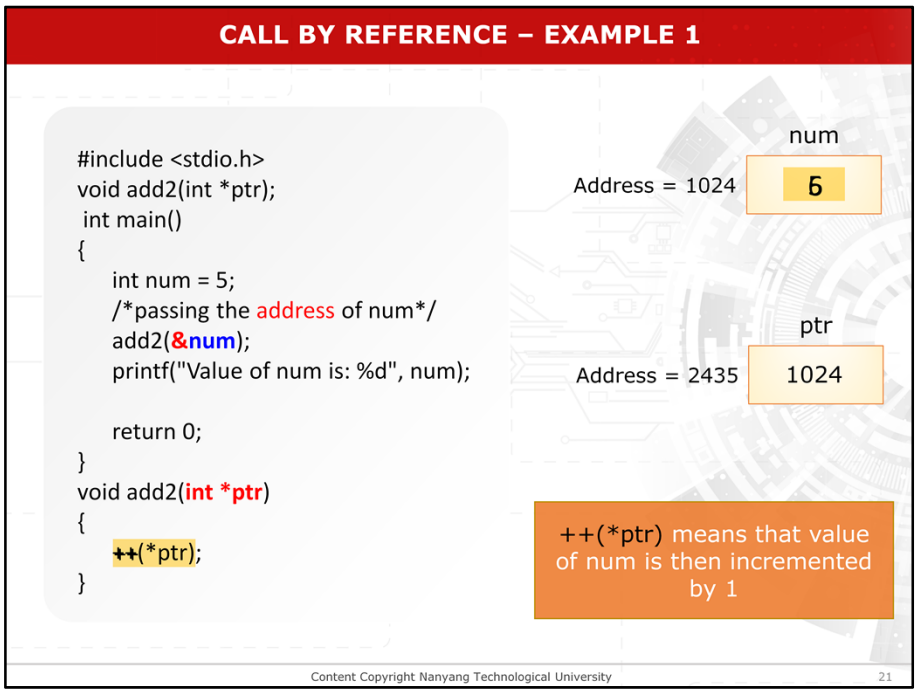
```
#include <stdio.h>
void add2(int *ptr);
 int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d", num);

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




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

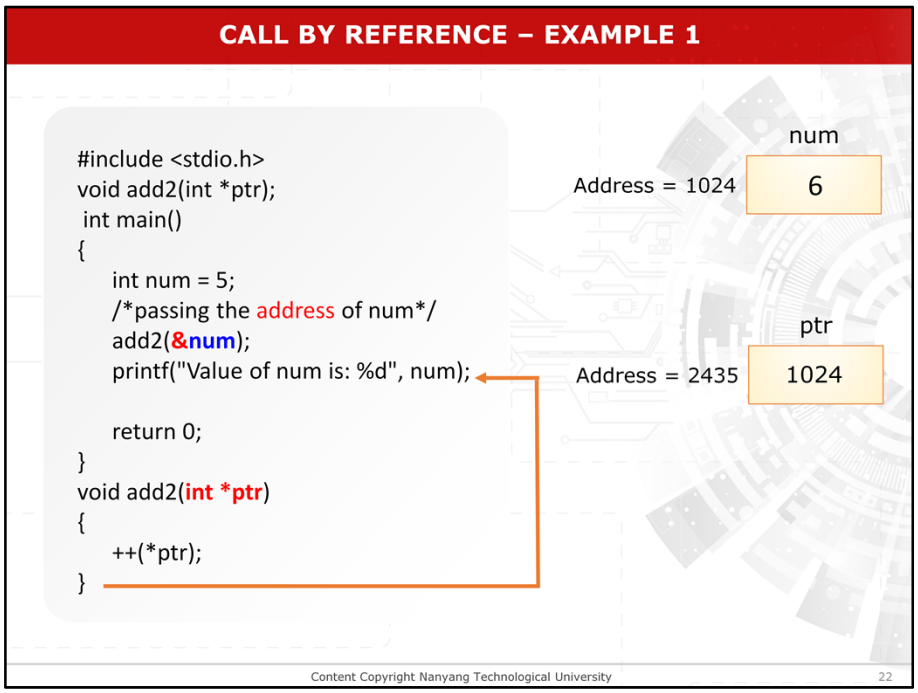



++(*ptr) means that the value of num is then incremented by 1



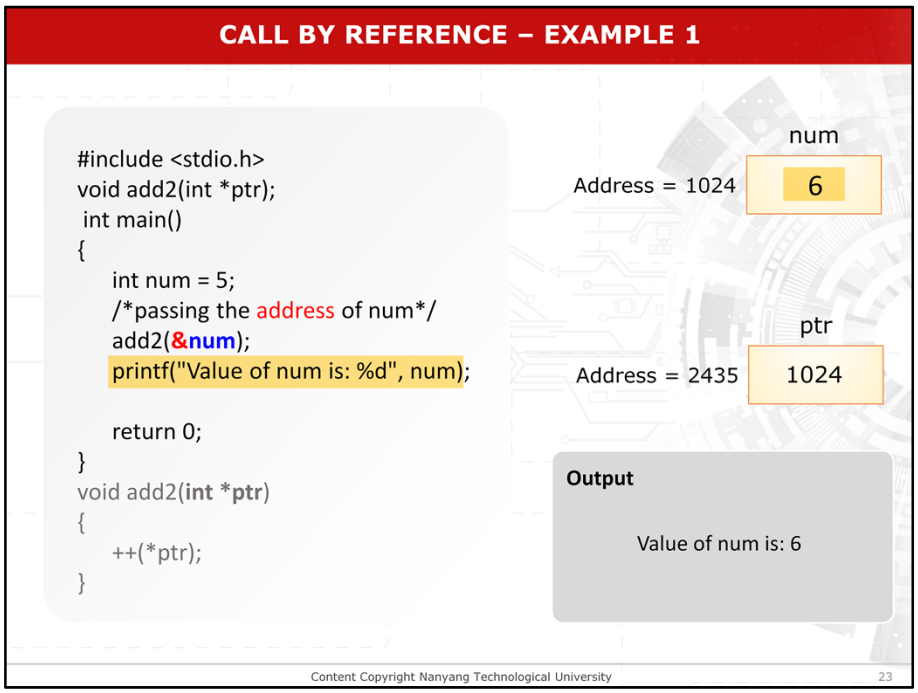

the value stored in the variable **num** increased by 1 from 5 to 6.

*[+animation]*

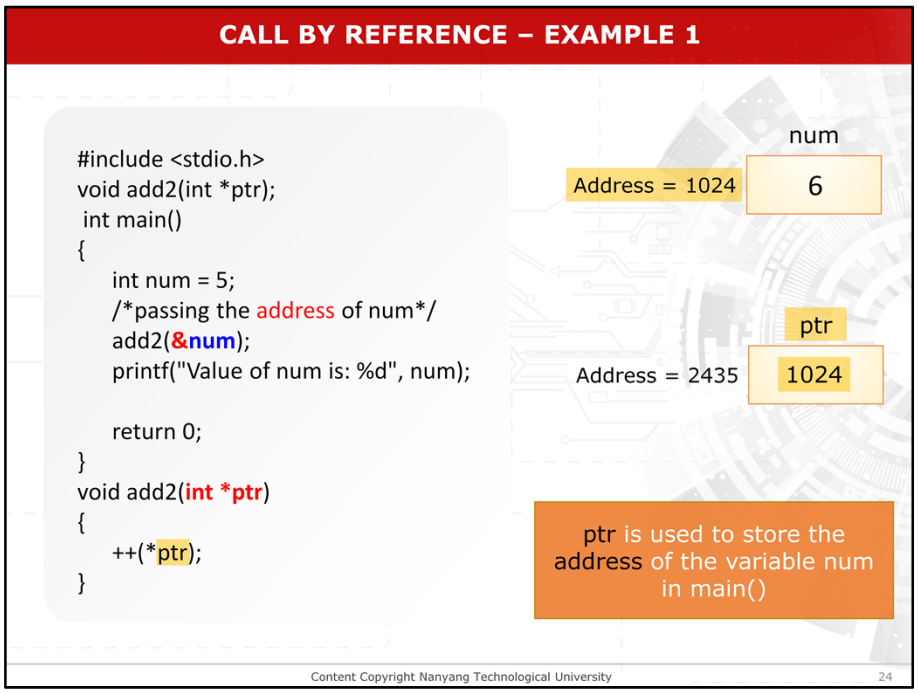



When the function ends, the control is then returned to the calling **main()** function.

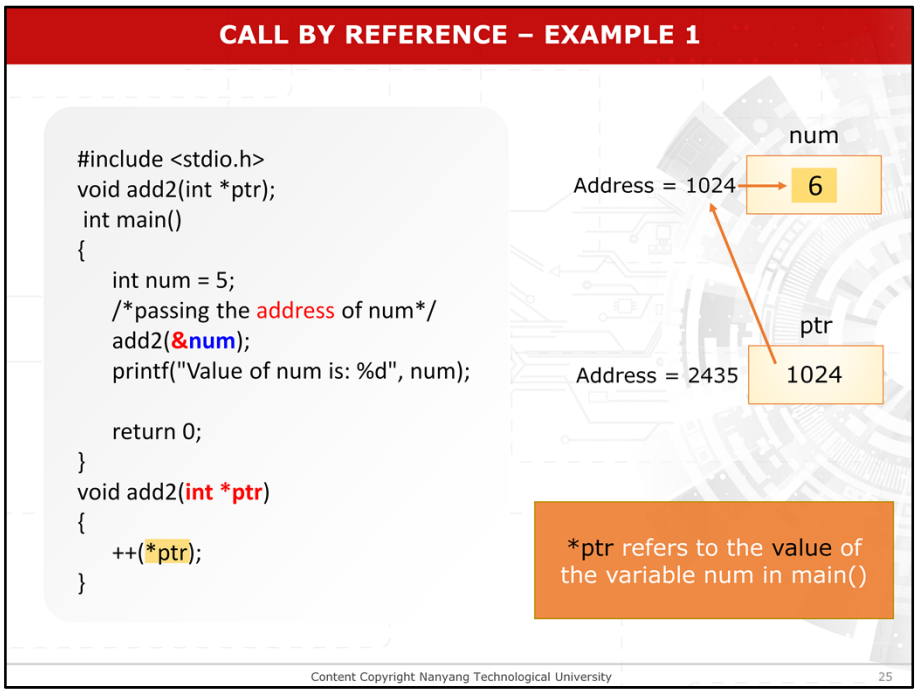
*[+animation]*




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



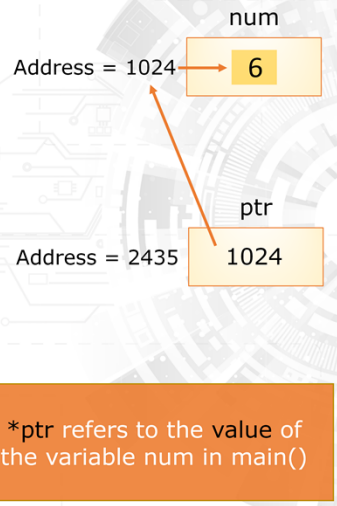

In this example, note that the parameter variable **ptr** in **add2()** is used to store the address of the variable **num** in **main()**.




*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);
}
```







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 examples 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;
}
```




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;
}
```




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;
}
```




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;
}
```




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




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

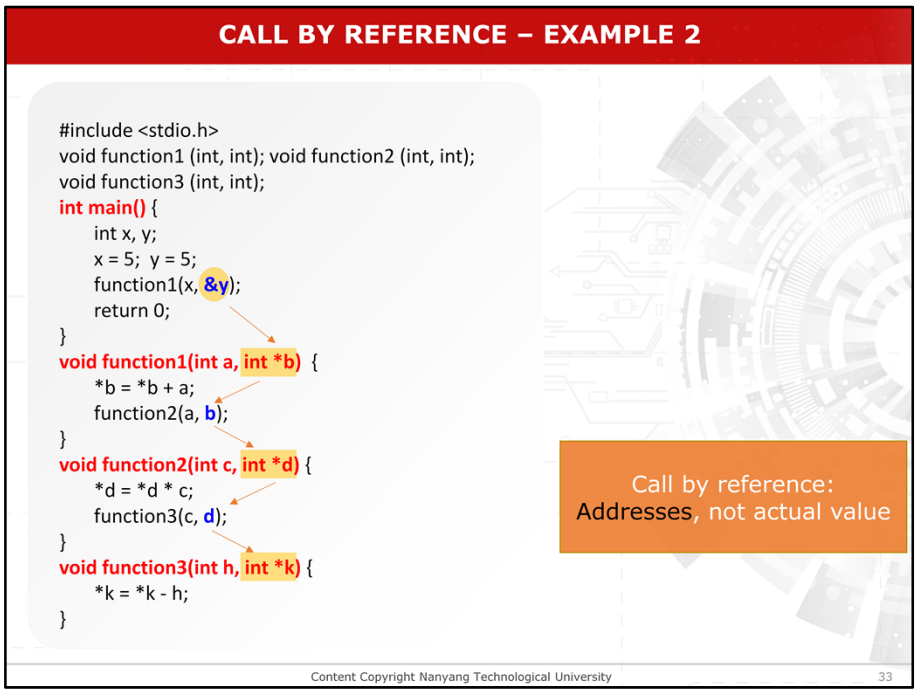
## CALL BY REFERENCE – EXAMPLE 2

```
#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




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




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]*

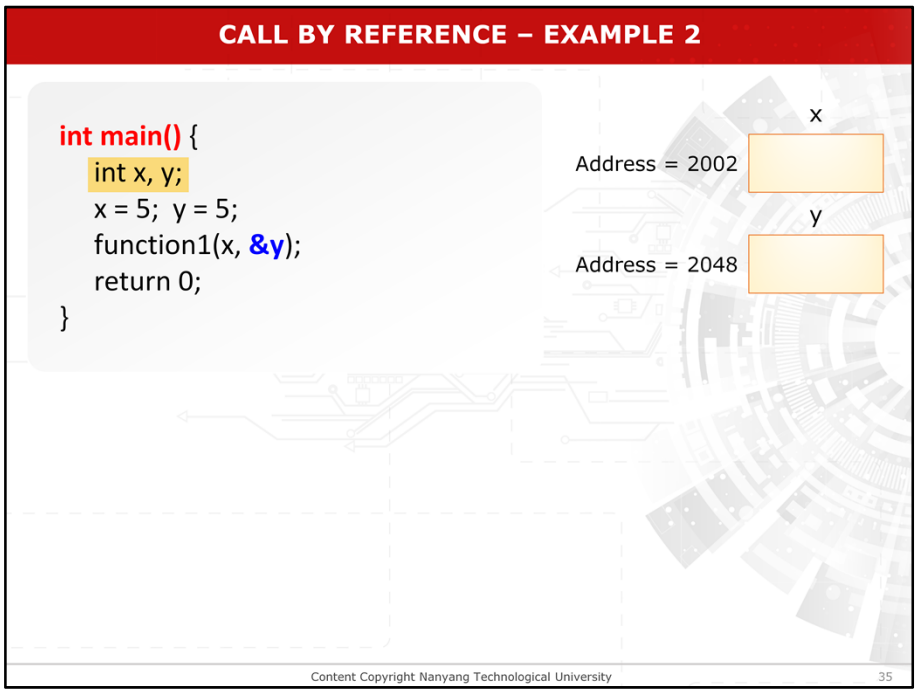
We will look into each part of the coding in detail.

## 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;
}
```

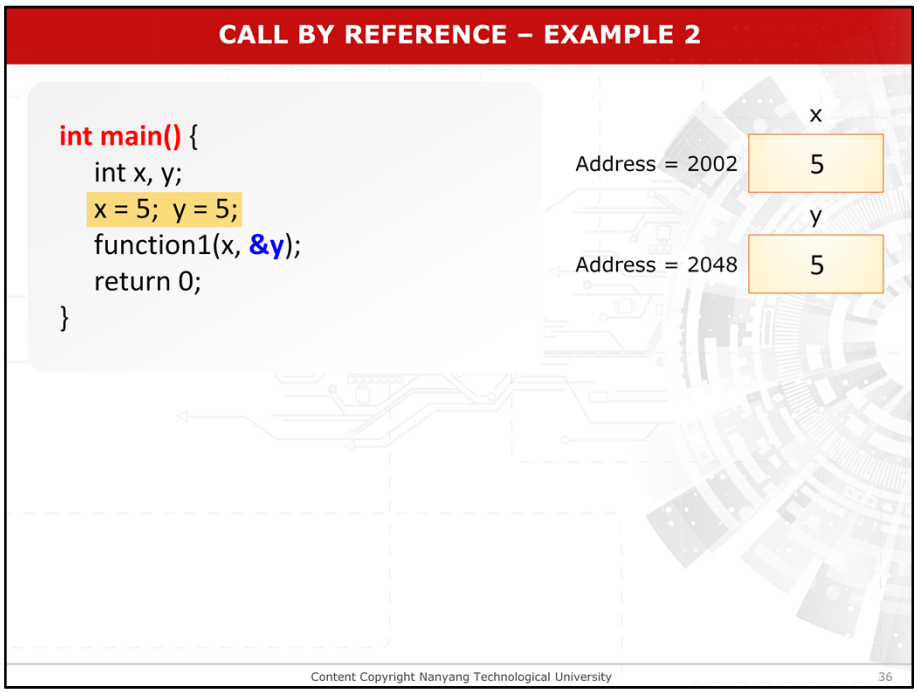



Let's start with the main() function.



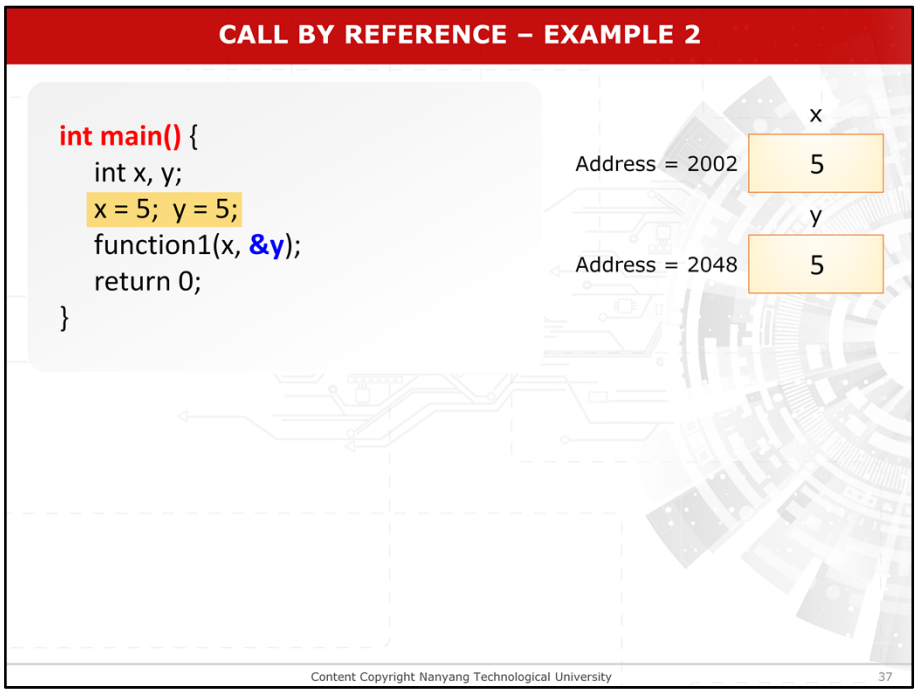

When the program starts execution, in the **main()** function, memory locations are allocated for the variables **x** and **y** accordingly.

*[+animations]*




The two variables are assigned with the value of 5.

*[+animations]*




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

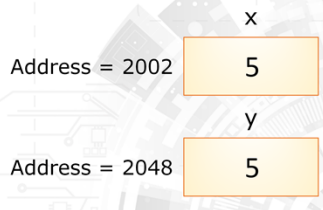
| | x | y | a | *b | c | *d | h | *k |
|---|---|---|---|---|---|---|---|---|
| (i) | 5 | **5** | - | - | - | - | - | - |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |




The declared values of x and y are 5.

## CALL BY REFERENCE – EXAMPLE 2

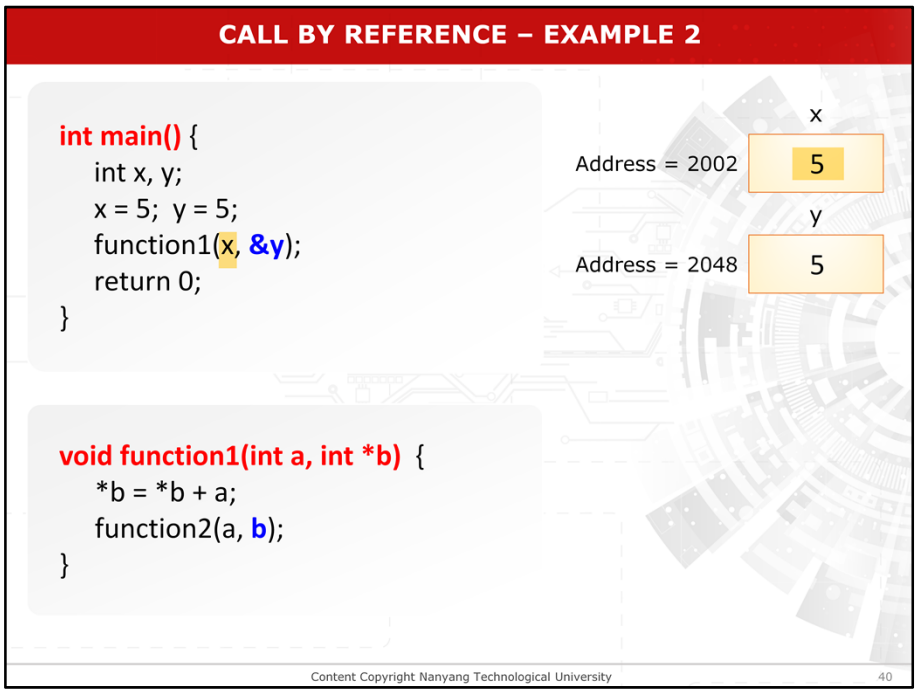
```
int main() {
    int x, y;
    x = 5;  y = 5;
    function1(x, &y);
    return 0;
}
```



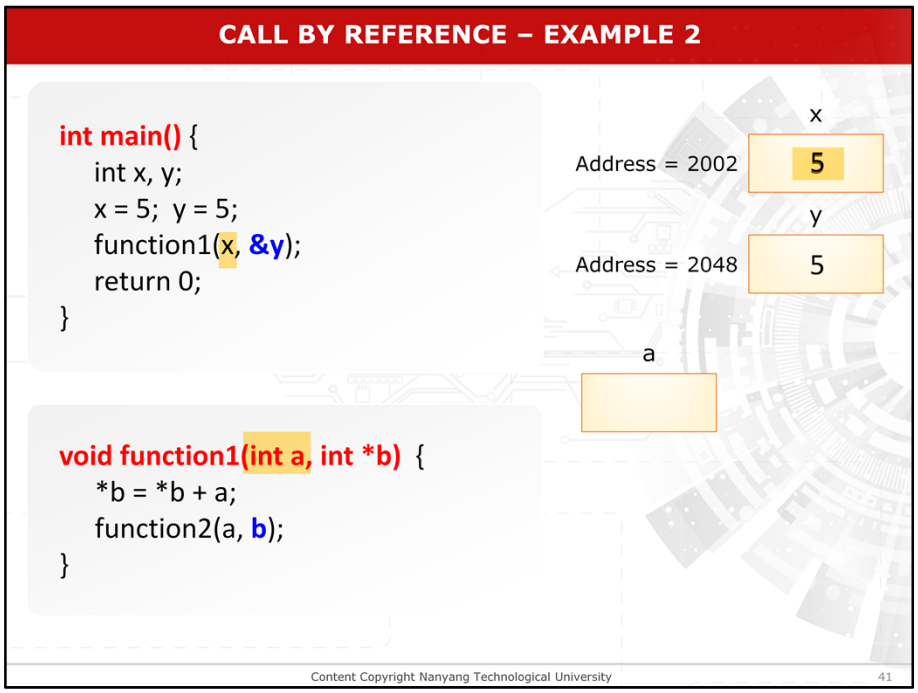




The **main()** function then calls **function1()**

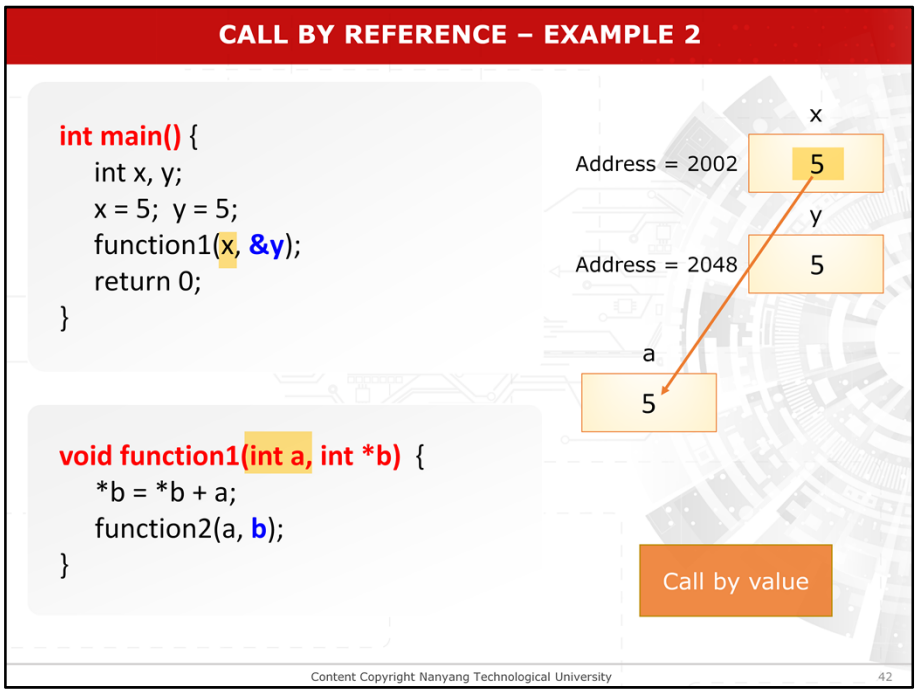



by passing the value of **x** (i.e. 5)

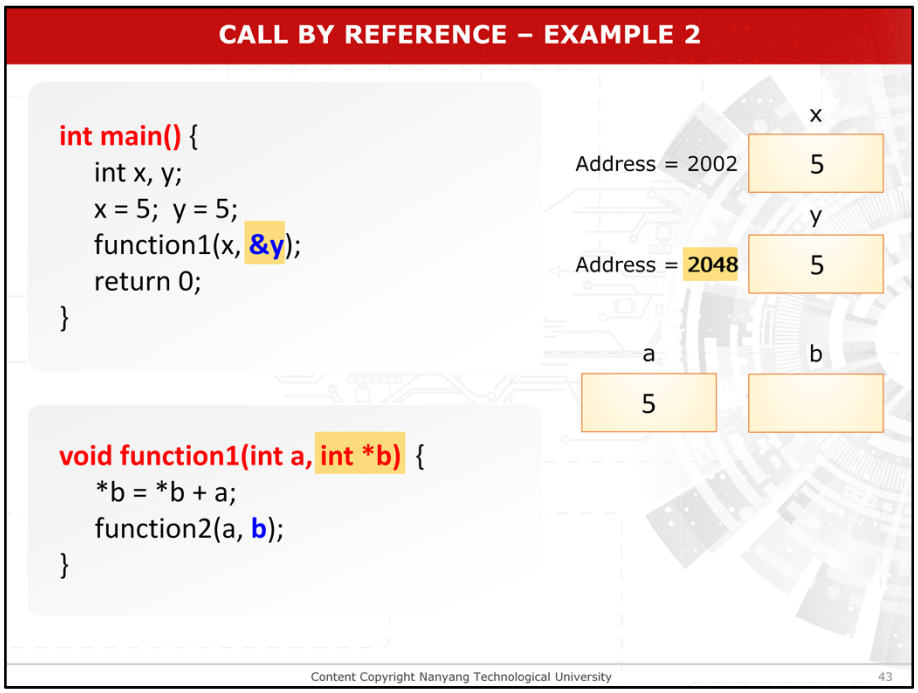



To parameters **a**

*[+animations]*

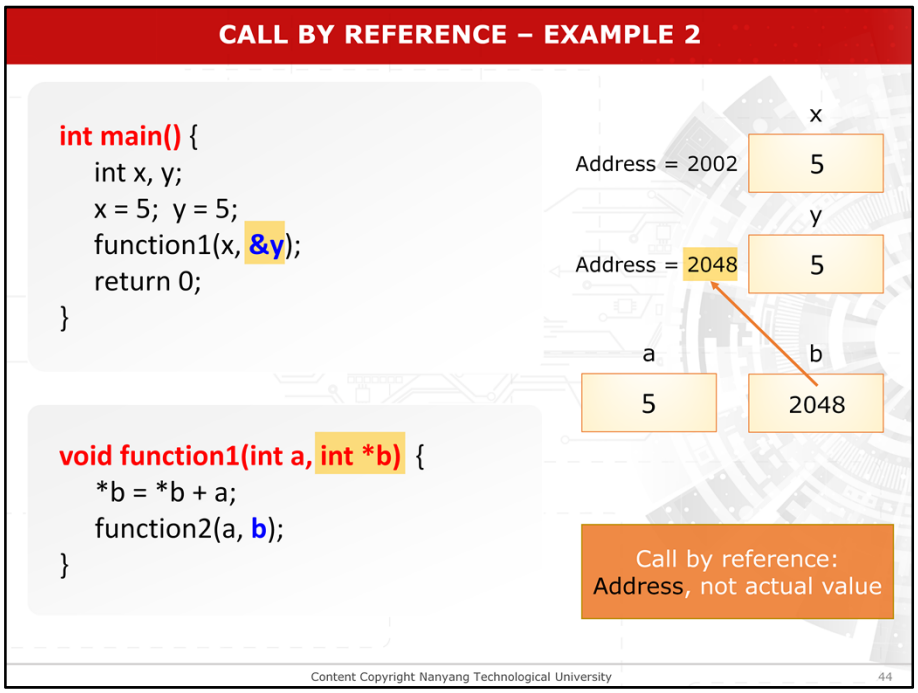



The mode of parameter passing for **a** is call by value

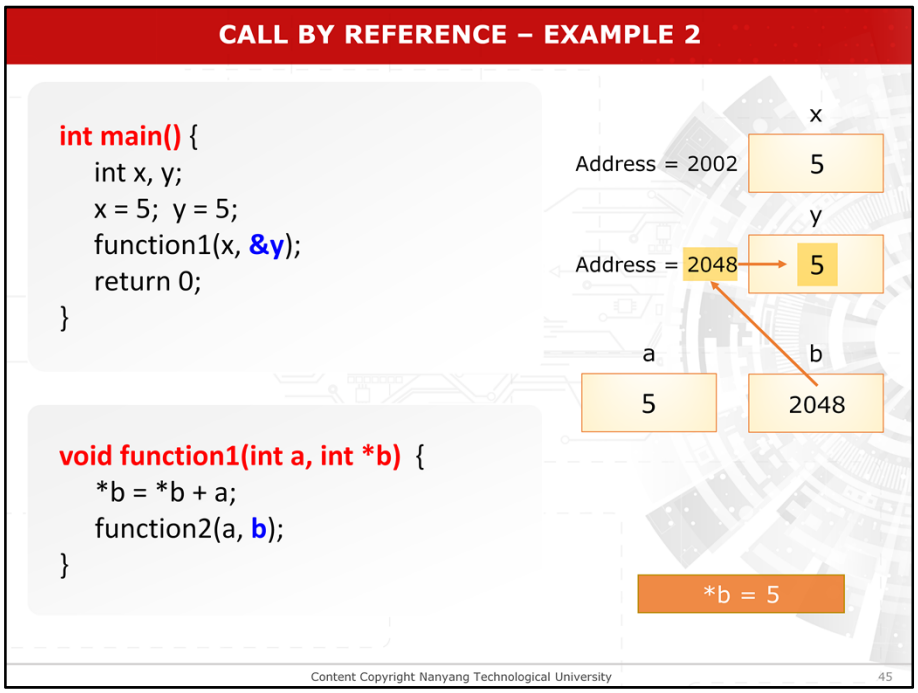



The address of **y** (i.e. 2048) is passed to parameter **b**.

*[+animations]*




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.




The value of *b is 5.

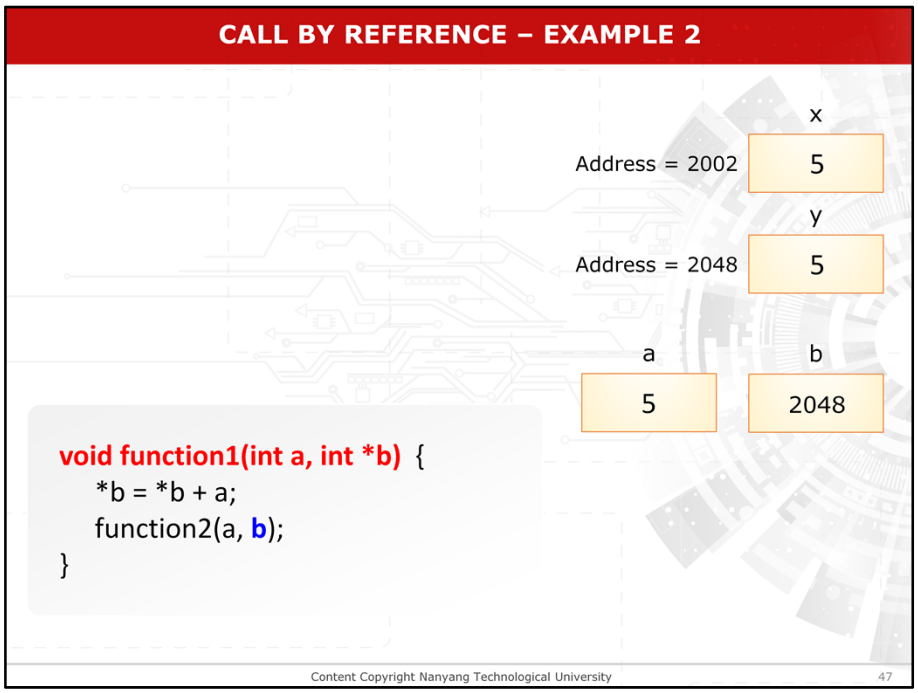
## CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

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

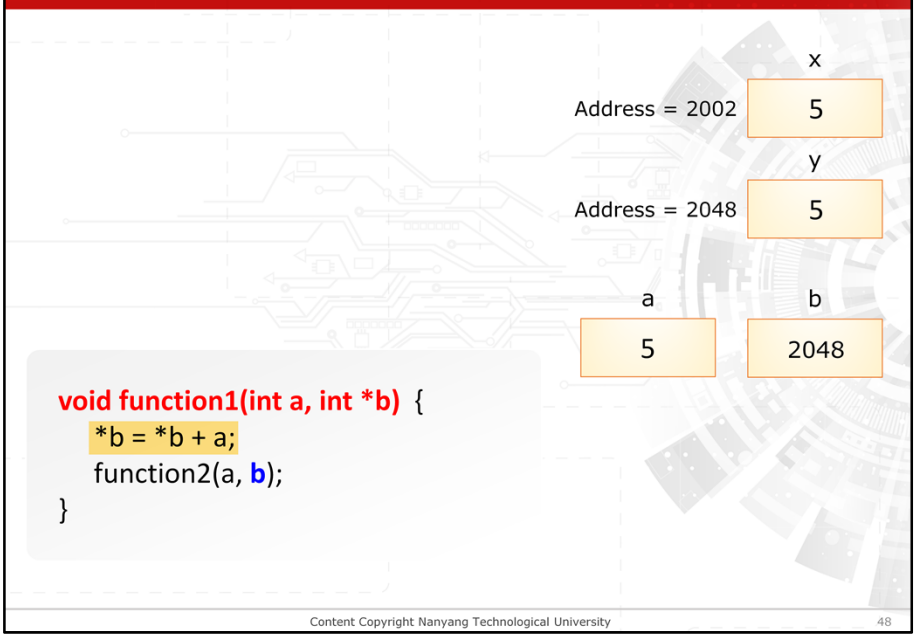



When function1 is called, values of x, y, a and *b are all 5.




When **function1()** is executed,

## CALL BY REFERENCE – EXAMPLE 2

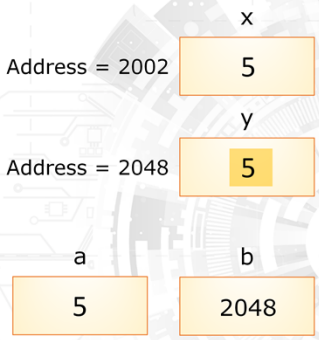






the statement ***b = *b + a** will update the value of

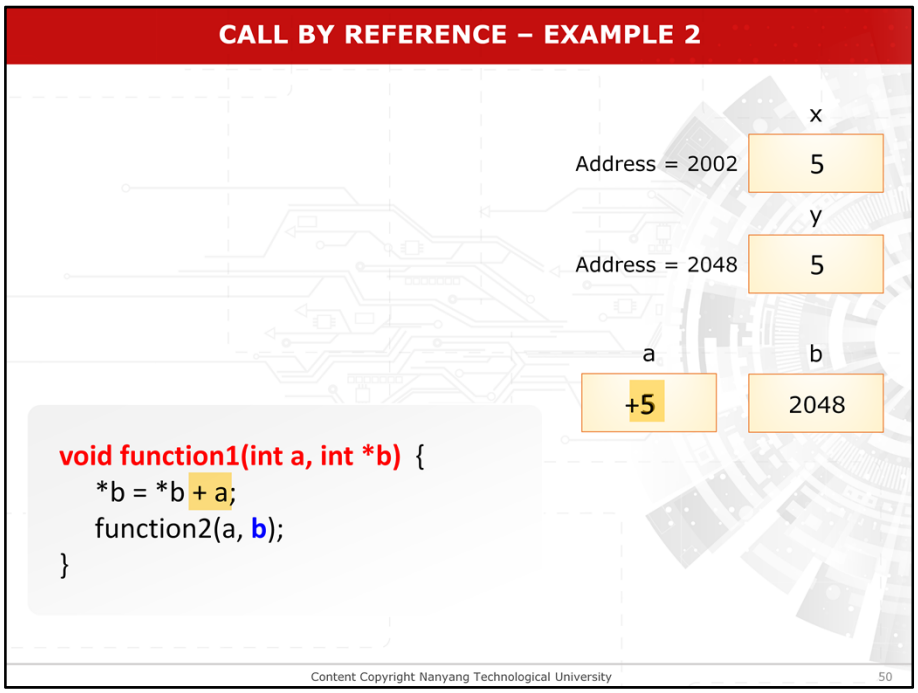
## CALL BY REFERENCE – EXAMPLE 2




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



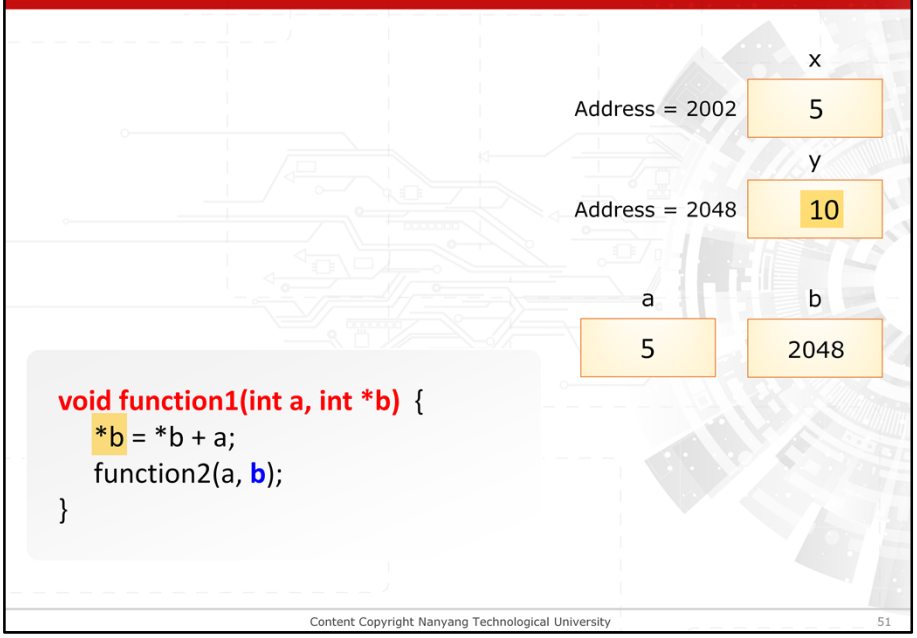

***b** = 5




+ 5

*[+animations]*

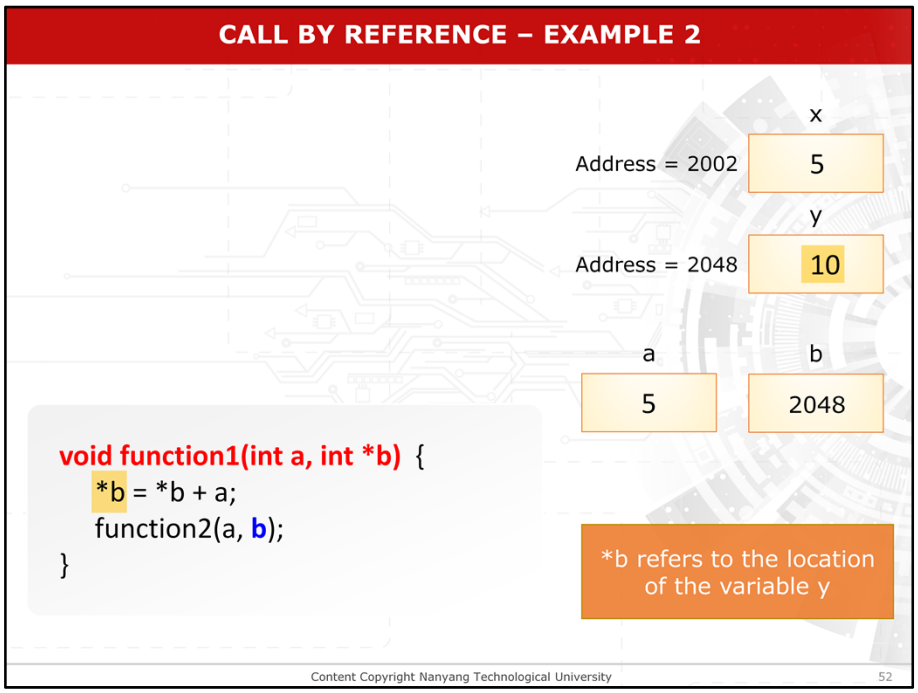
## CALL BY REFERENCE – EXAMPLE 2



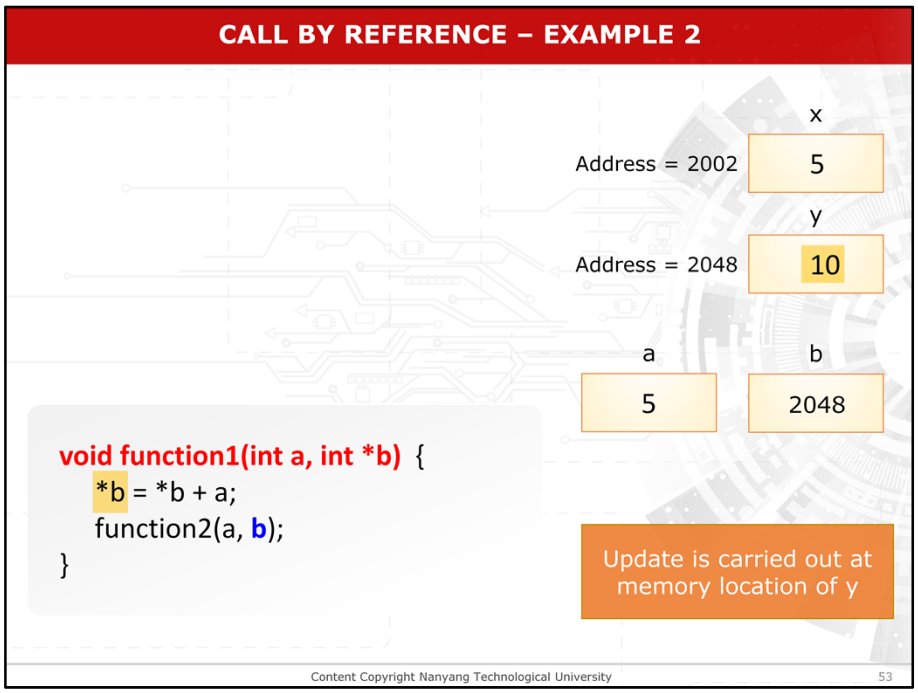

```
void function1(int a, int *b) {
    *b = *b + a;
    function2(a, b);
}
```



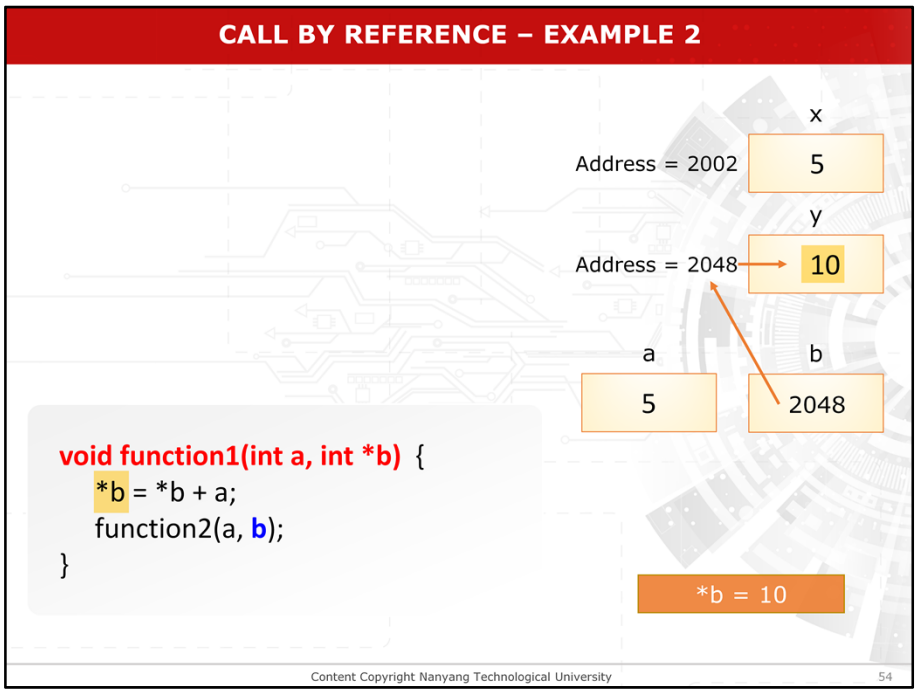

= 10




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




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.

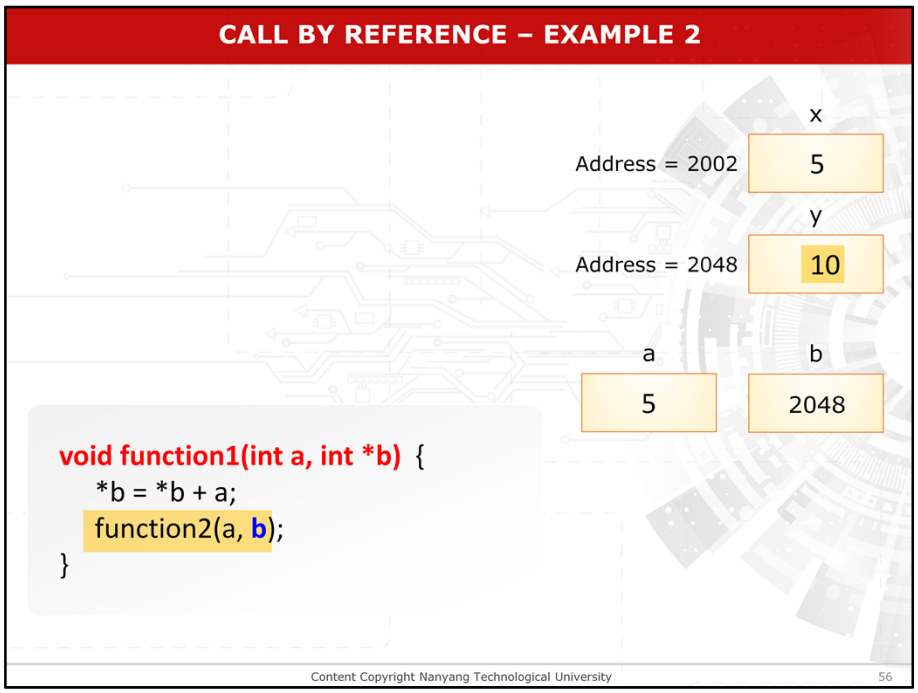
## CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

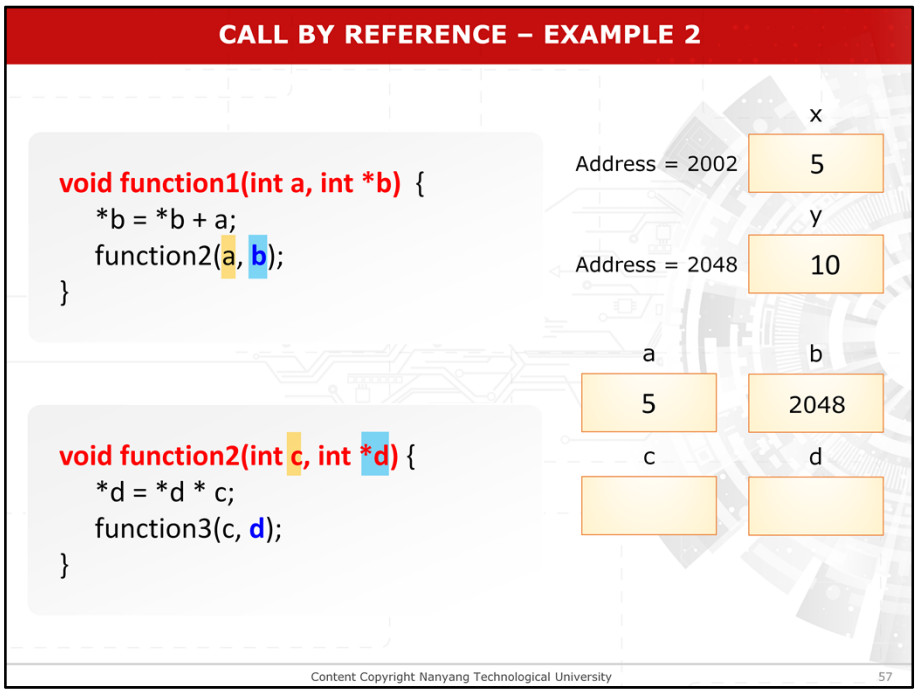
| | x | y | a | *b | c | *d | h | *k |
|---|---|---|---|---|---|---|---|---|
| (i) | 5 | **5** | - | - | - | - | - | - |
| (ii) | 5 | **5** | 5 | **5** | - | - | - | - |
| (iii) | 5 | **10** | 5 | **10** | - | - | - | - |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |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




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

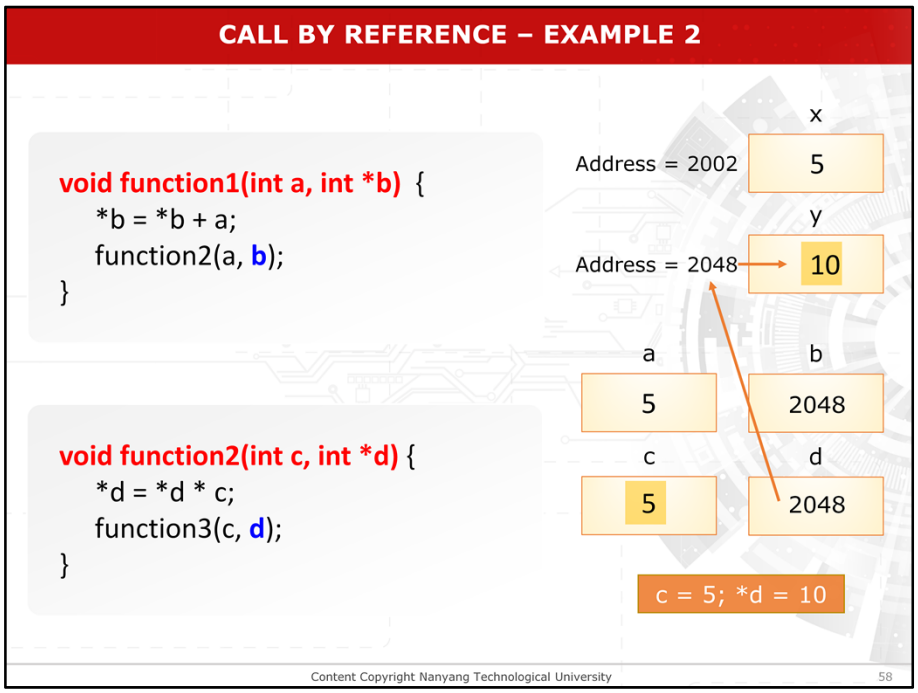



After that, **function1()** calls **function2()**




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

*[+animations]*




The values of c is 5 and *d is 10

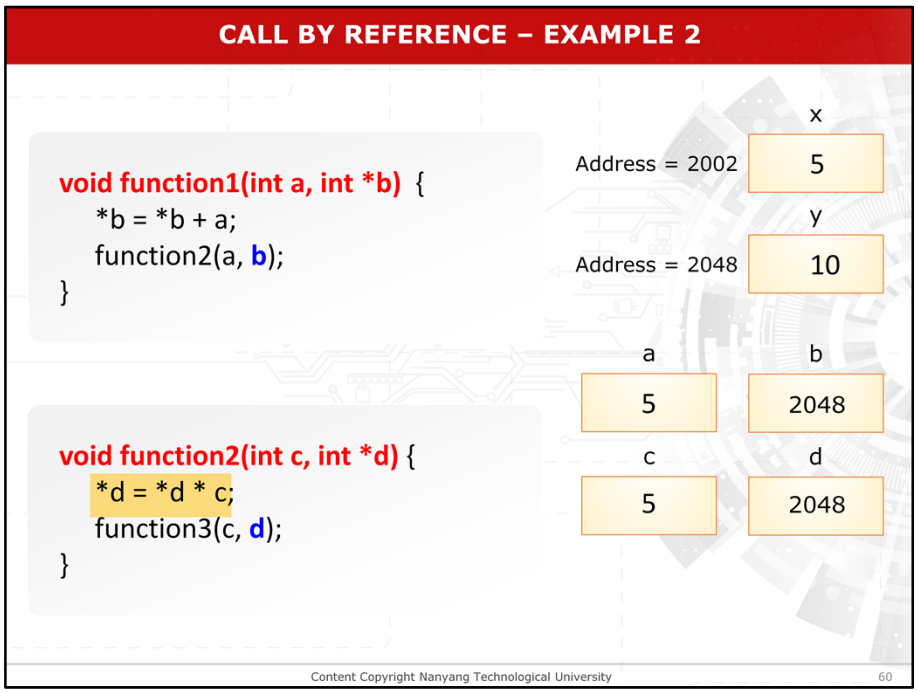
## CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

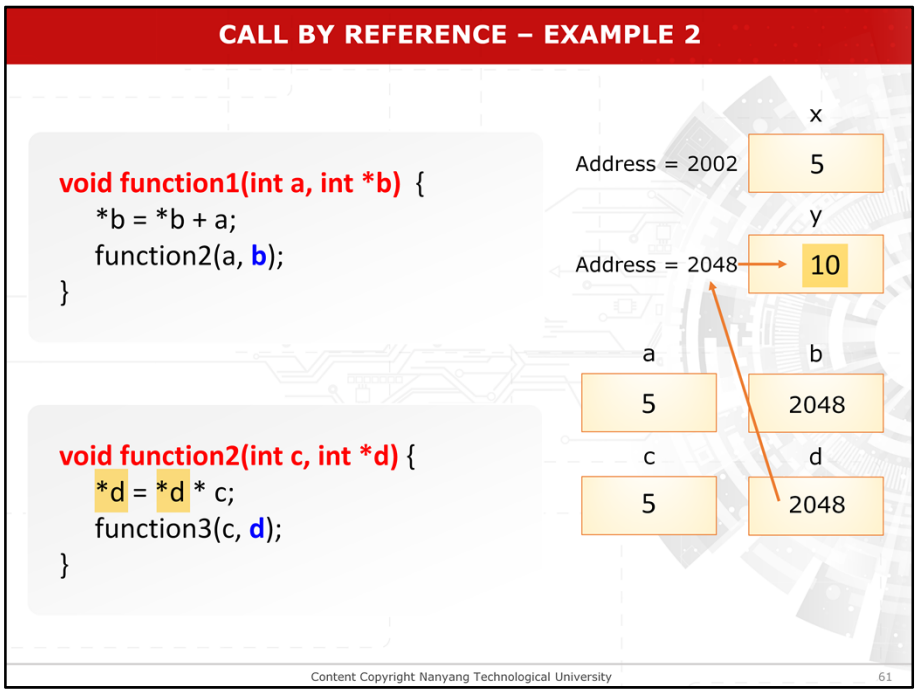
| | x | y | a | *b | c | *d | h | *k |
|---|---|---|---|---|---|---|---|---|
| (i) | 5 | **5** | - | - | - | - | - | - |
| (ii) | 5 | **5** | 5 | **5** | - | - | - | - |
| (iii) | 5 | **10** | 5 | **10** | - | - | - | - |
| (iv) | 5 | **10** | 5 | **10** | 5 | **10** | - | - |
| | | | | | | | | |
| |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



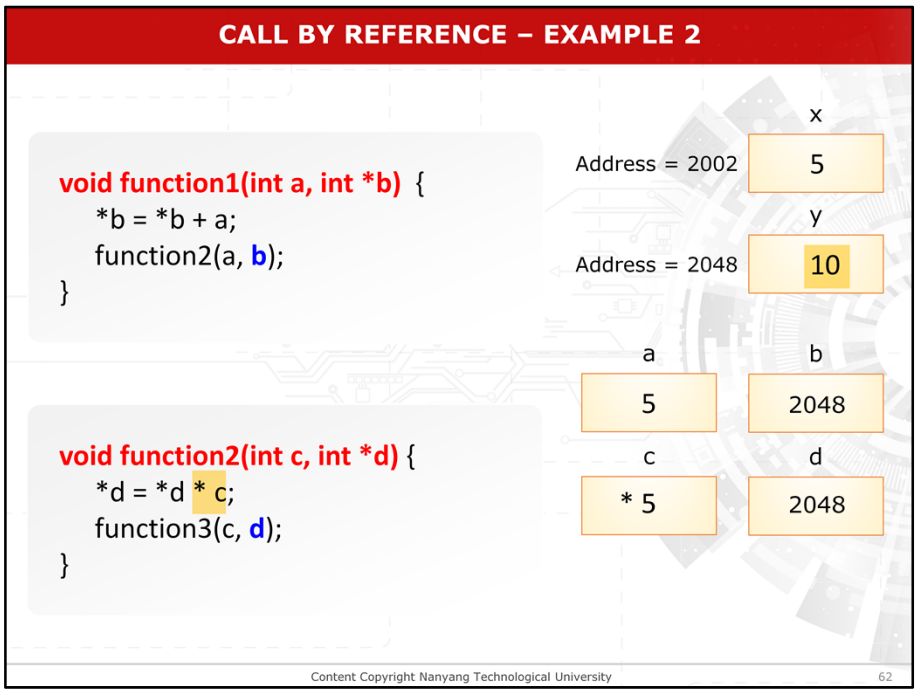

The tracking table is updated as shown.




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

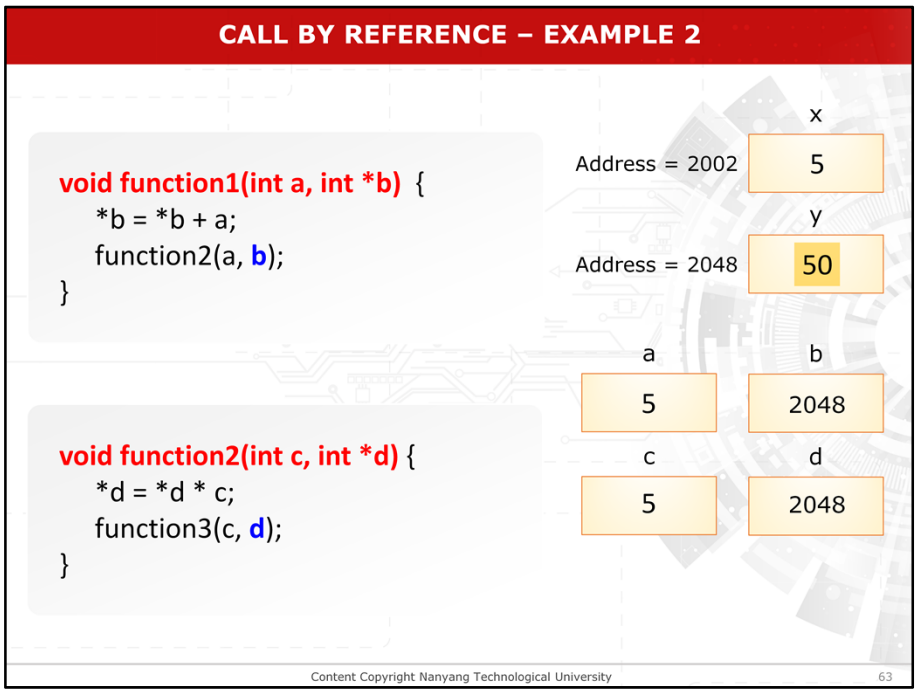

CALL BY REFERENCE – EXAMPLE 2
void function1(int a, int *b) {
*b = *b + a;
function2(a, b);
}
void function2(int c, int *d) {
*d = *d * c;
function3(c, d);
}
x
Address = 2002
5
y
Address = 2048
10
a
5
b
2048
c
5
d
2048
Content Copyright Nanyang Technological University
61

***d** = 10




Multiply by 5

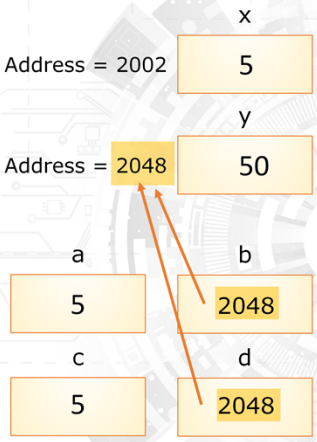
*[+animations]*


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
Address = 2002
5
y
Address = 2048
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;
}
```



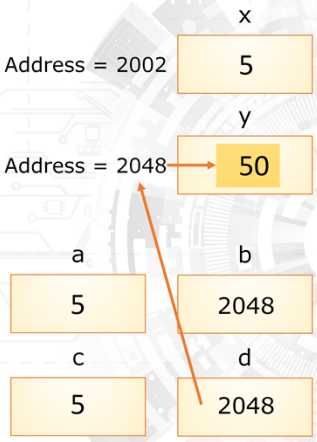




As the pointer variable **d** contains the same address value as **b** in **function1()**, it also refers to the location of the variable **y** in the **main()** function.

## 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;
}
```







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.

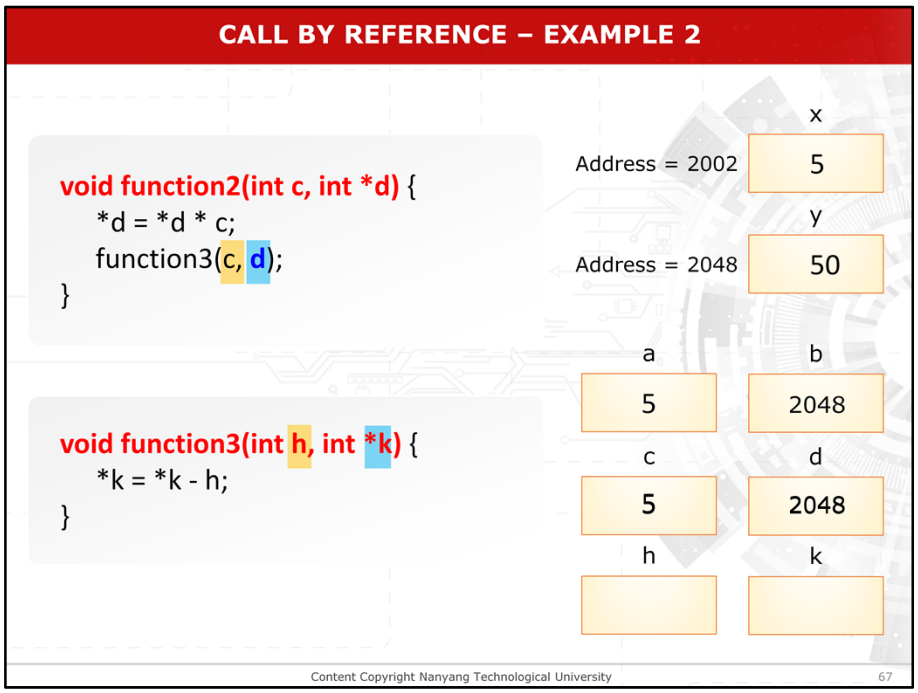
## CALL BY REFERENCE – EXAMPLE 2

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** | - | - |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

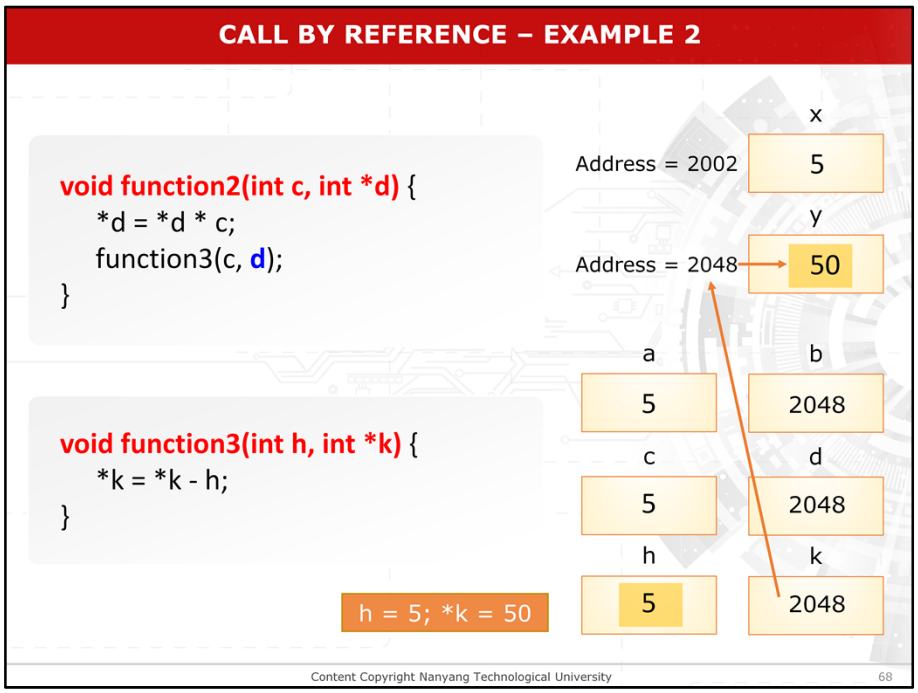



Let's update the tracking table as shown.




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

*[+animations]*




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

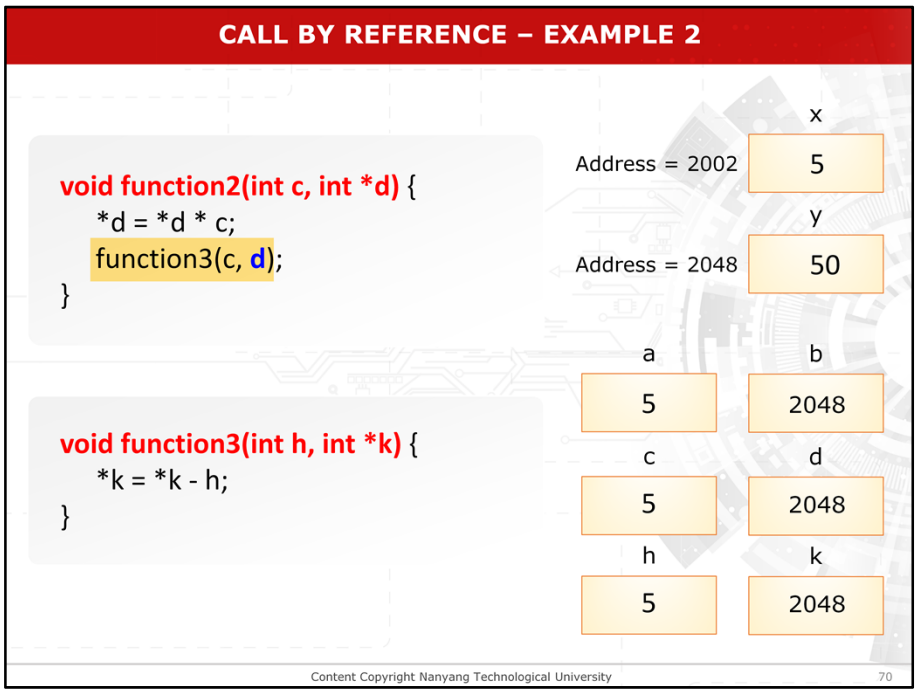
## CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

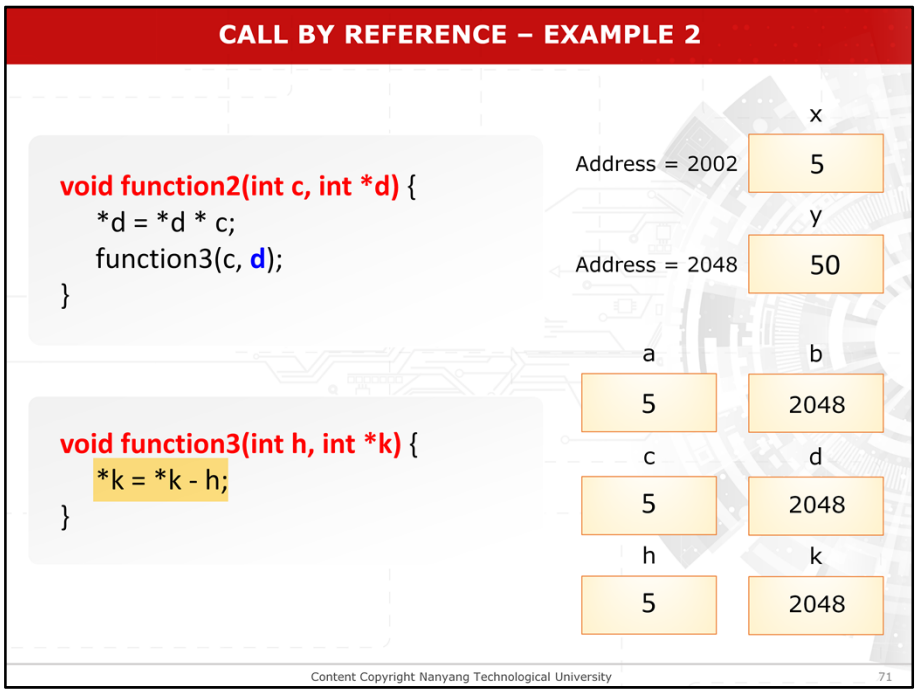
| | x | y | a | *b | c | *d | h | *k |
|---|---|---|---|---|---|---|---|---|
| (i) | 5 | **5** | - | - | - | - | - | - |
| (ii) | 5 | **5** | 5 | **5** | - | - | - | - |
| (iii) | 5 | **10** | 5 | **10** | - | - | - | - |
| (iv) | 5 | **10** | 5 | **10** | 5 | **10** | - | - |
| (v) | 5 | **50** | 5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



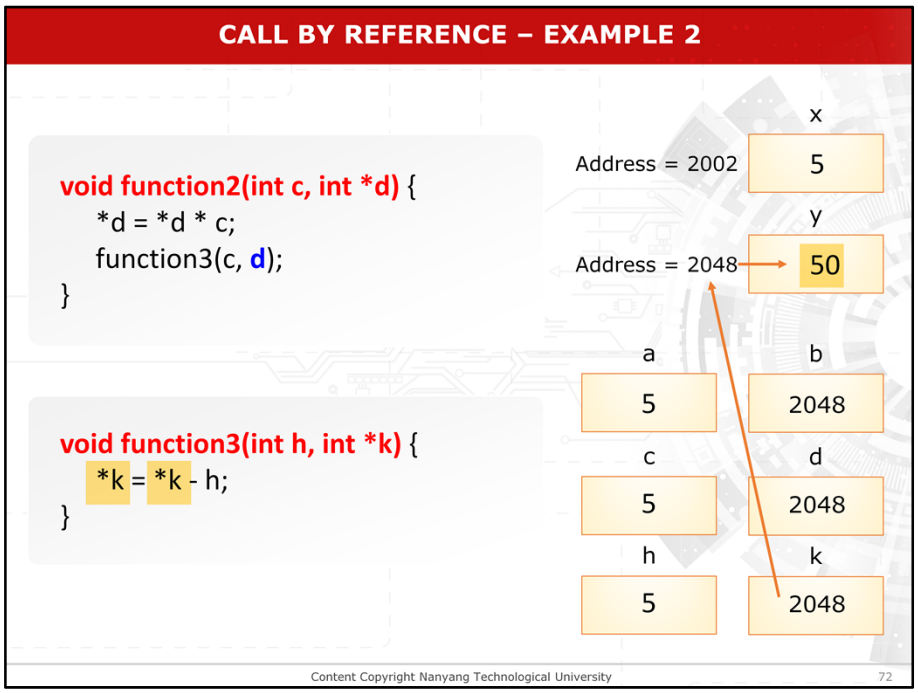

The tracking table is updated as shown.



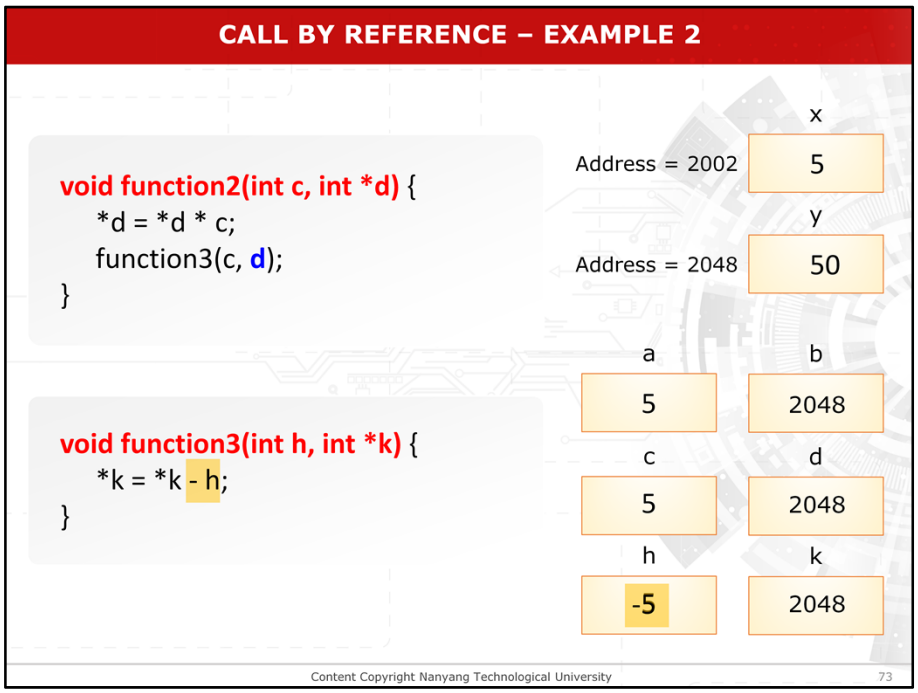

When **function3()** is executed,




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

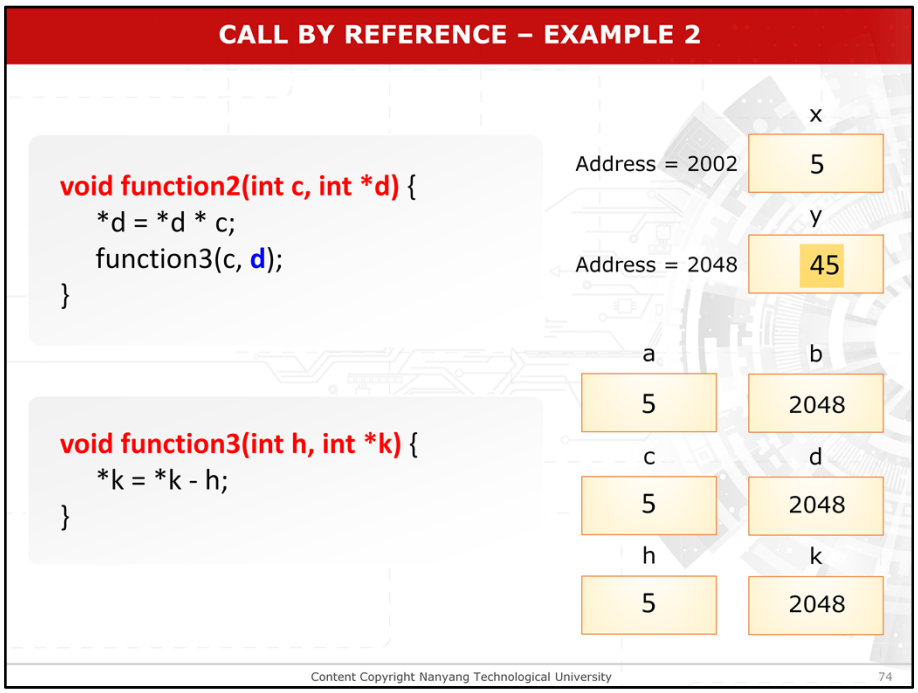

CALL BY REFERENCE – EXAMPLE 2
void function2(int c, int *d) {
*d = *d * c;
function3(c, d);
}
void function3(int h, int *k) {
*k = *k - h;
}
x
Address = 2002
5
y
Address = 2048
50
a
5
b
2048
c
5
d
2048
h
5
k
2048
Content Copyright Nanyang Technological University
72

value of ***k** = 50




-5

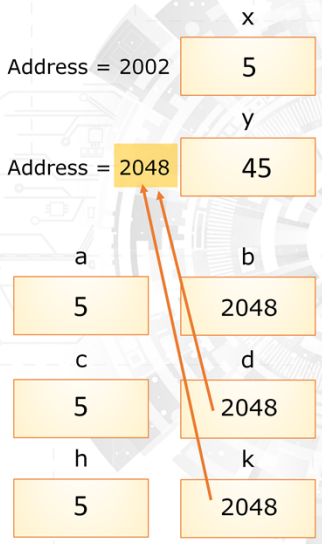
*[+animations]*


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
Address = 2002
5
y
Address = 2048
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;
}
```





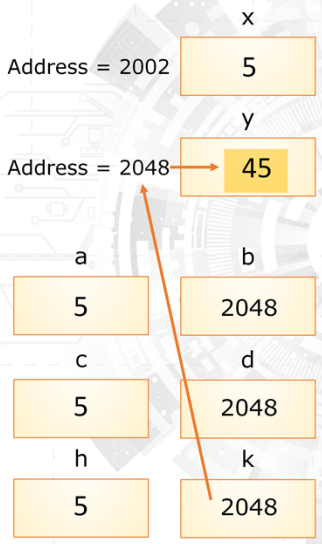


As the pointer variable **k** contains the same address value as **d** in **function2()**, it also refers to the location of the variable **y** in the **main()** function.

## 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







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.

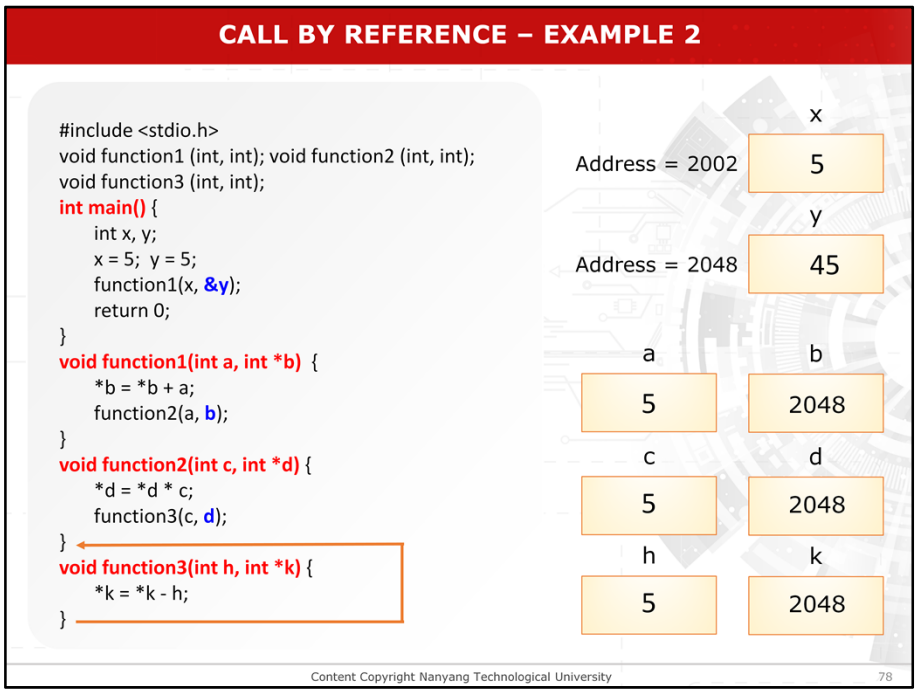
## CALL BY REFERENCE – EXAMPLE 2

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** |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

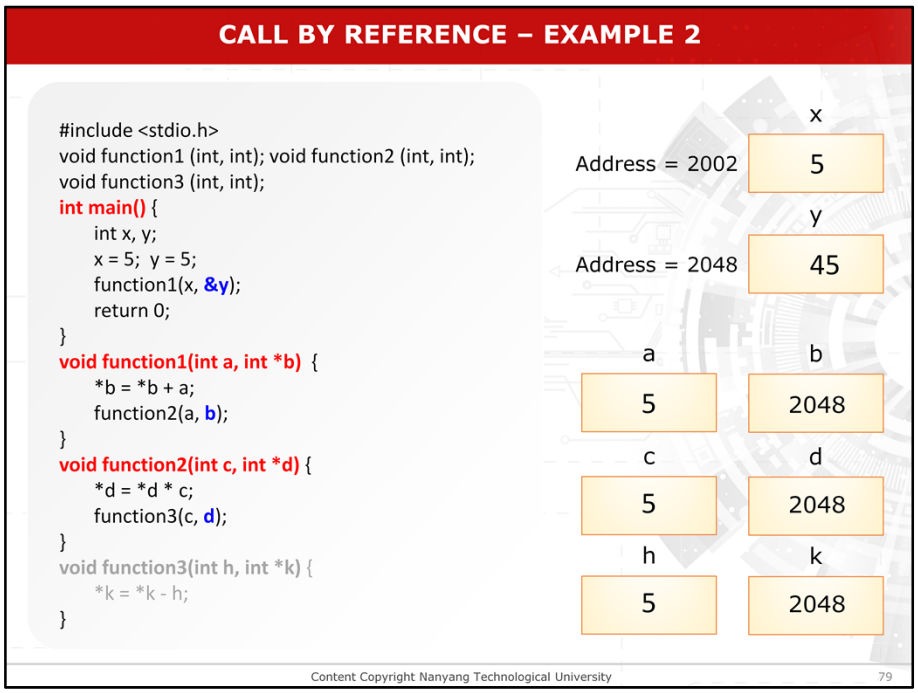



The tracking table is updated as shown.




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

*[+animations]*




Parameters h and k are released.

*[+animations]*

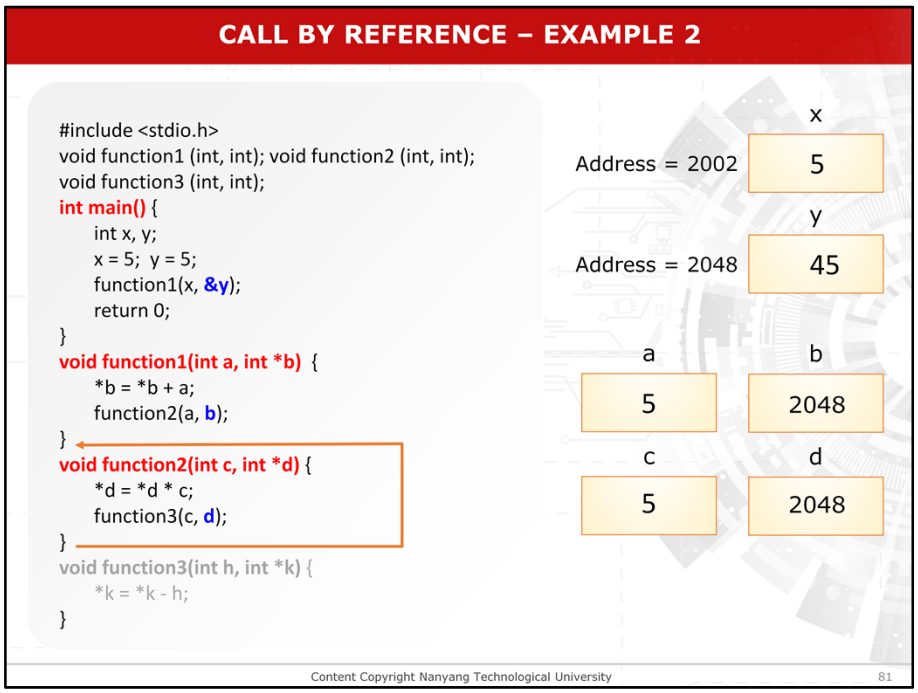
## CALL BY REFERENCE – EXAMPLE 2

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** | - | - |
| | | | | | | | | |
| | | | | | | | | |

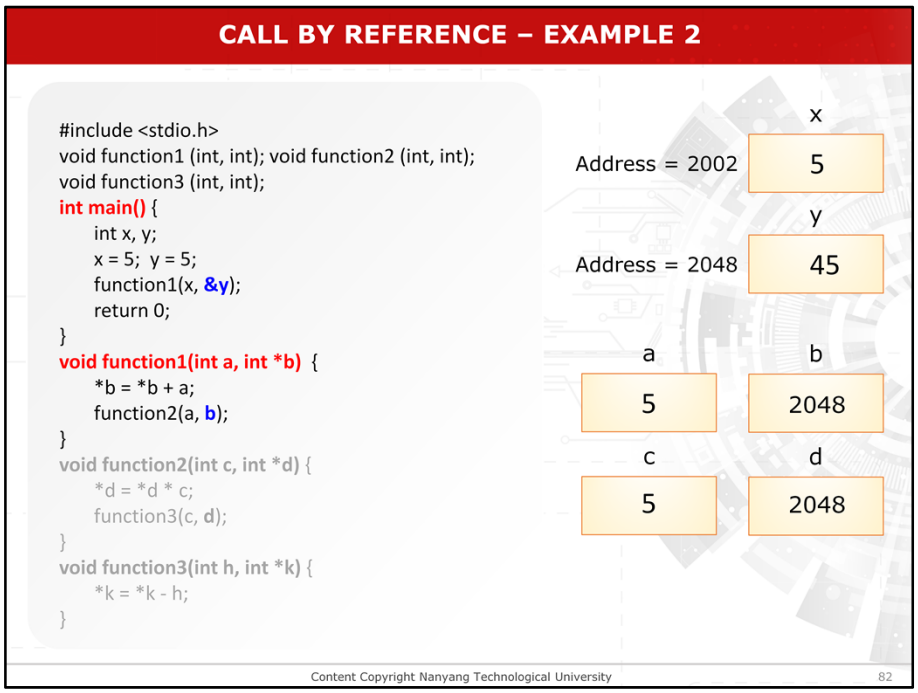



The tracking table is updated as shown.




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

*[+animations]*




Parameters c and d are released

*[+animations]*

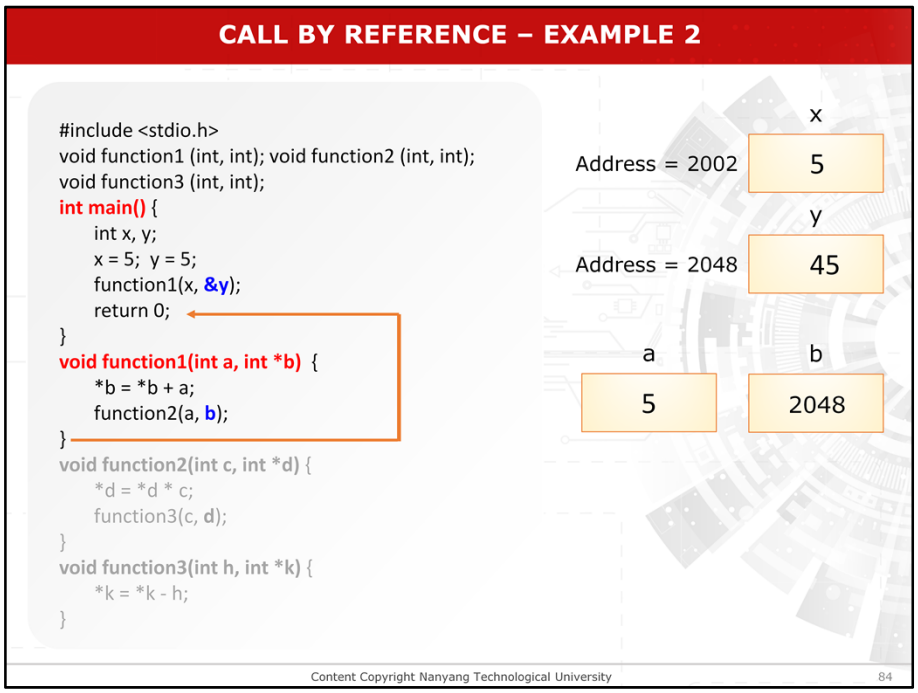
## CALL BY REFERENCE – EXAMPLE 2

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** | - | - | - | - |
| | | | | | | | | |

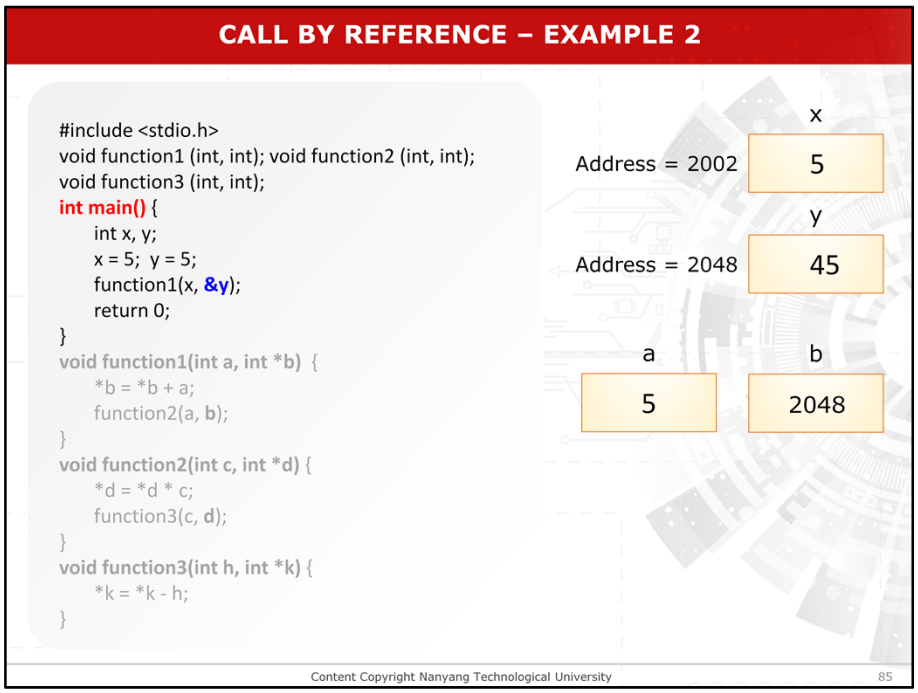



The tracking table is updated as shown.




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

*[+animations]*




Parameter a and b are released.

*[+animations]*

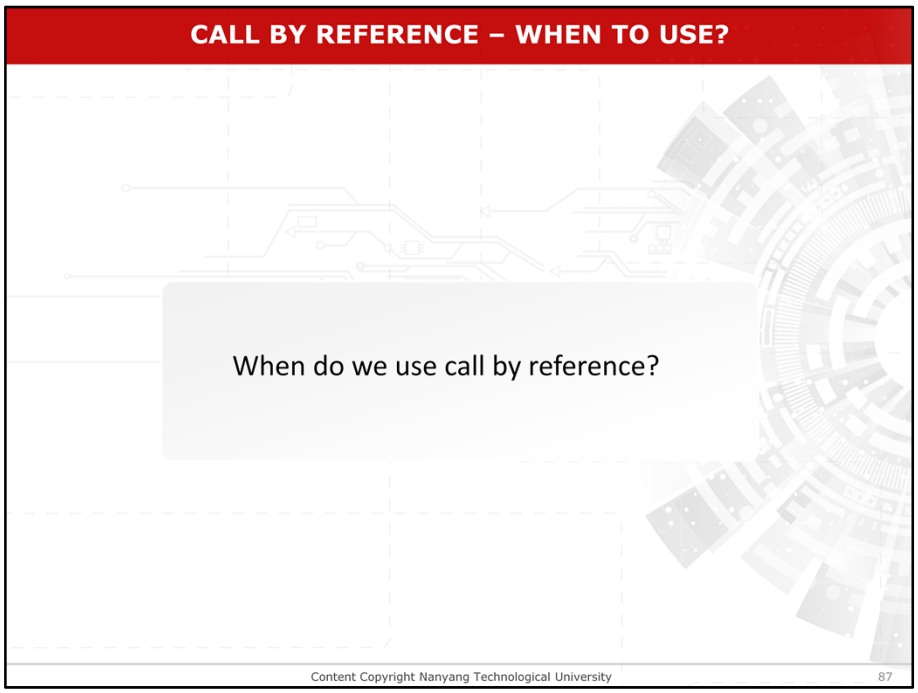
## CALL BY REFERENCE – EXAMPLE 2

Tracking of the changes in the parameters

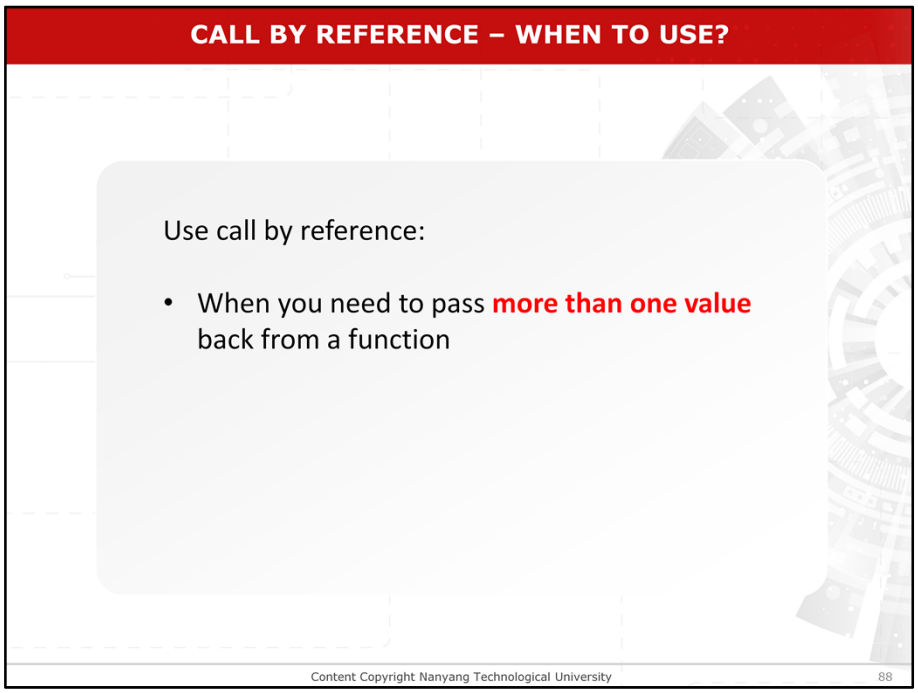
| | x | y | a | *b | c | *d | h | *k |
|---|---|---|---|---|---|---|---|---|
| (i) | 5 | **5** | - | - | - | - | - | - |
| (ii) | 5 | **5** | 5 | **5** | - | - | - | - |
| (iii) | 5 | **10** | 5 | **10** | - | - | - | - |
| (iv) | 5 | **10** | 5 | **10** | 5 | **10** | - | - |
| (v) | 5 | **50** | 5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



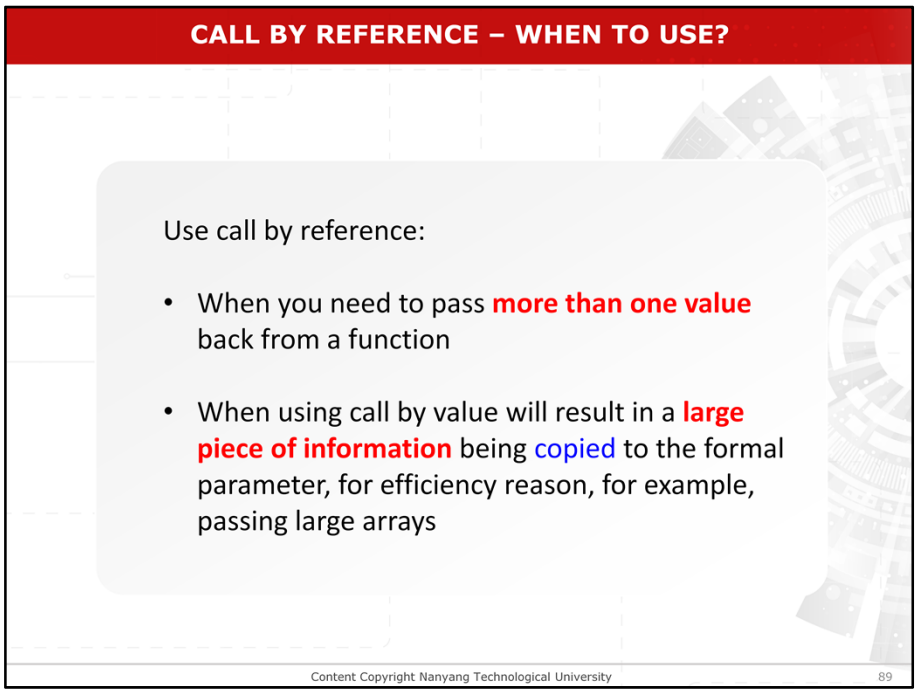

The tracking table is updated as shown. So from this example, we see the effect of using call by reference.




When do we use call by reference?




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




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;
}
```




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;
}
```





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;
}
```

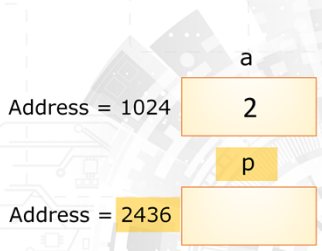




Let's assume the memory address for parameter a is 1024.

## 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;
}
```



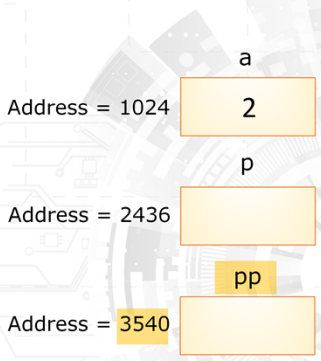




Next, let's assume parameter p was declared as a pointer variable at memory address 2436.

## DOUBLE INDIRECTION – EXAMPLE

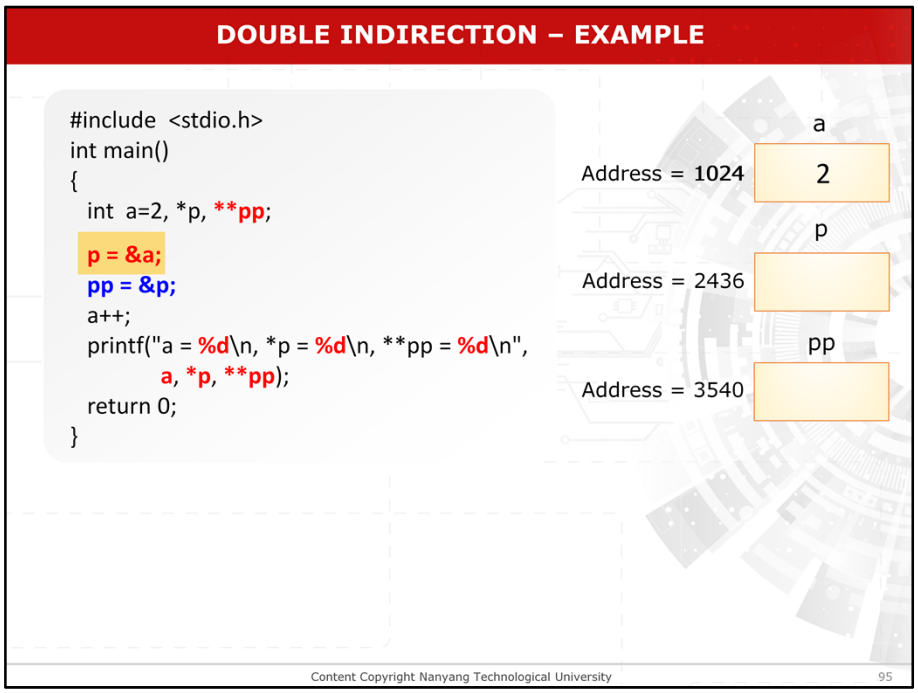
```
#include  <stdio.h>
int main()
{
  int  a=2, *p, **pp;

  p = &a;
  pp = &p;
  a++;
  printf("a = %d\n, *p = %d\n, **pp = %d\n",
          a, *p, **pp);
  return 0;
}
```







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

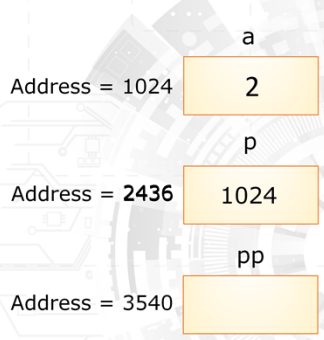



In the highlighted statement, the value of the address of a is stored in parameter p.

*[+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;
}
```

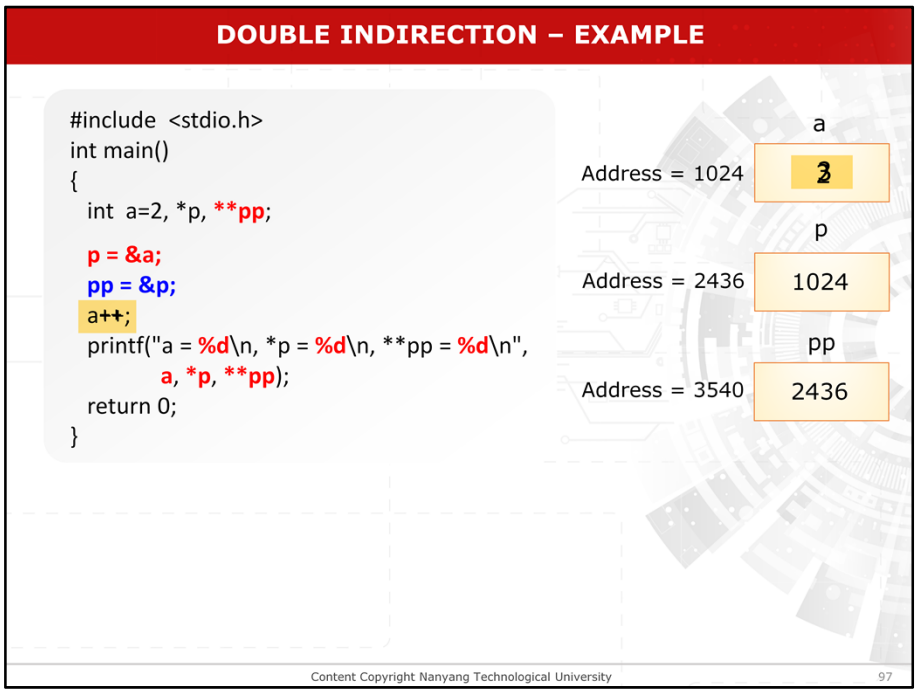






In the highlighted statement, the value of the address of p is stored in parameter pp.

*[+animations]*

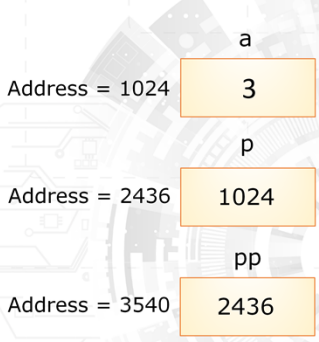



When a is increased by 1, the value of a changes from 2 to 3.

*[+animations]*

## DOUBLE INDIRECTION – EXAMPLE

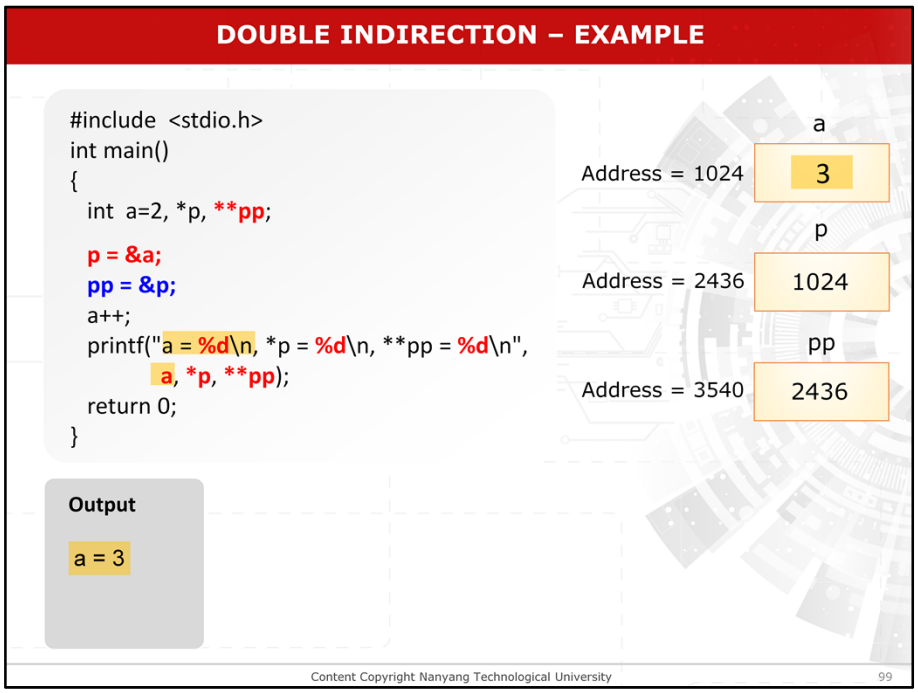
```
#include  <stdio.h>
int main()
{
  int  a=2, *p, **pp;

  p = &a;
  pp = &p;
  a++;
  printf("a = %d\n, *p = %d\n, **pp = %d\n",
          a, *p, **pp);
  return 0;
}
```



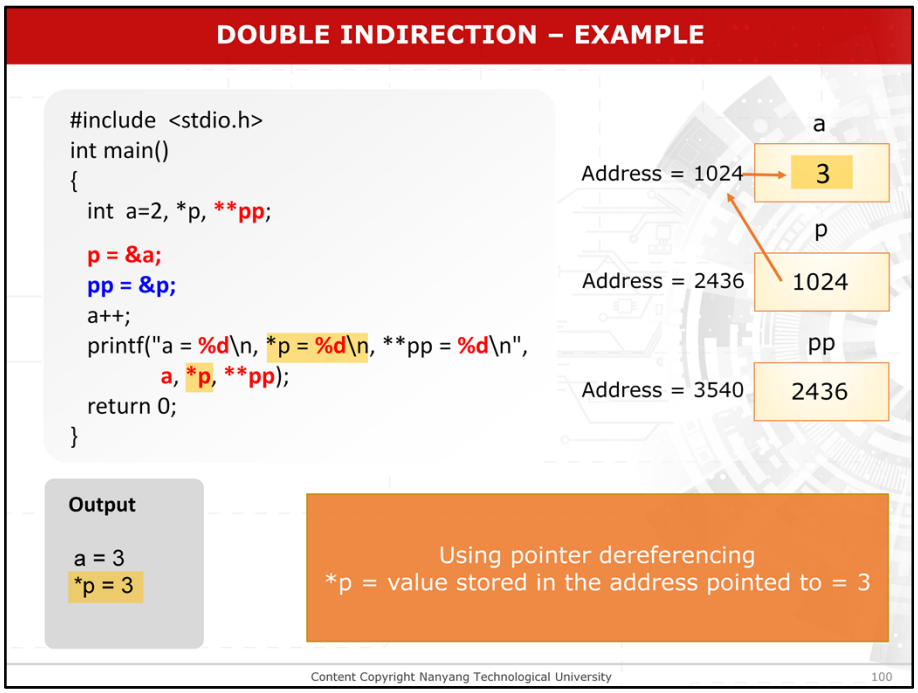

**Output**




Let's see what are the outputs.



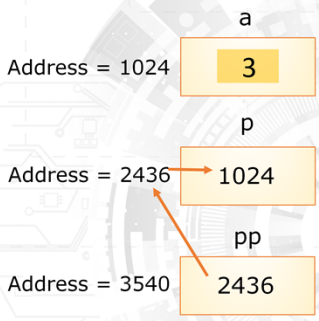

The value of a is 3.




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**

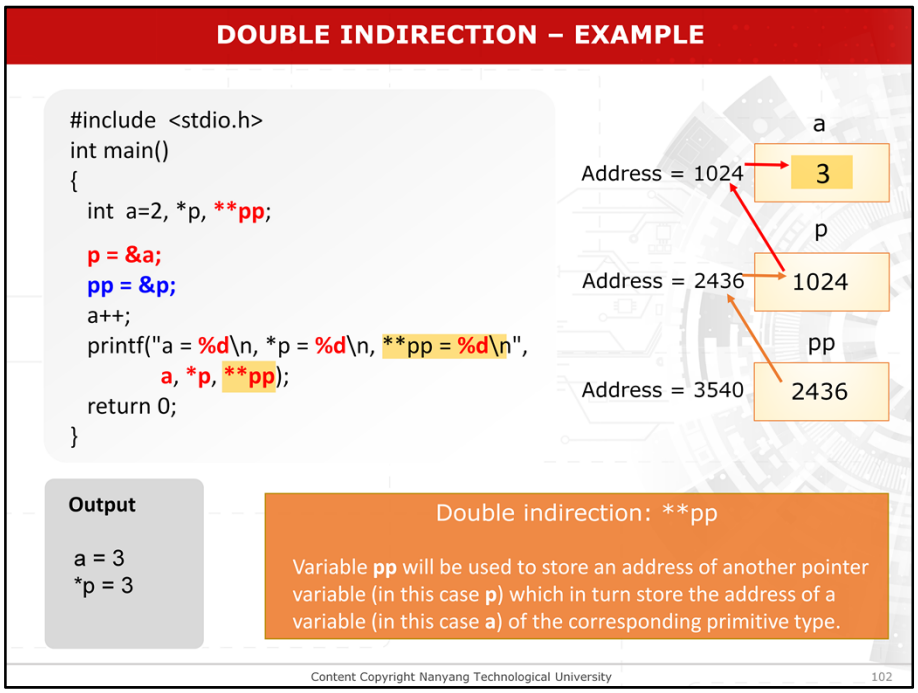
a = 3
*p = 3

**Double indirection: **pp**

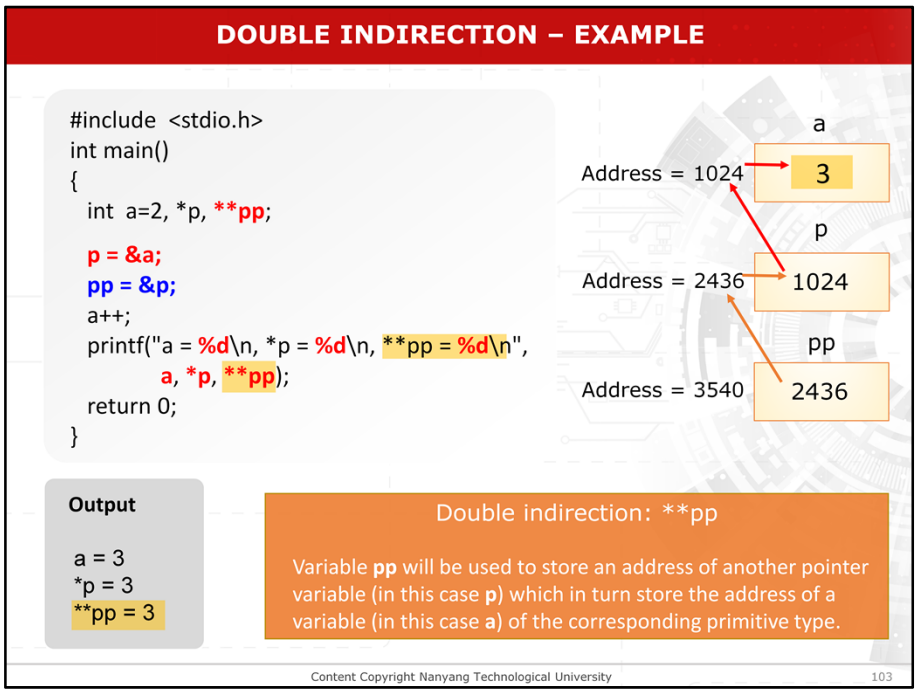
Variable **pp** will be used to store an address of another pointer variable (in this case **p**)




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

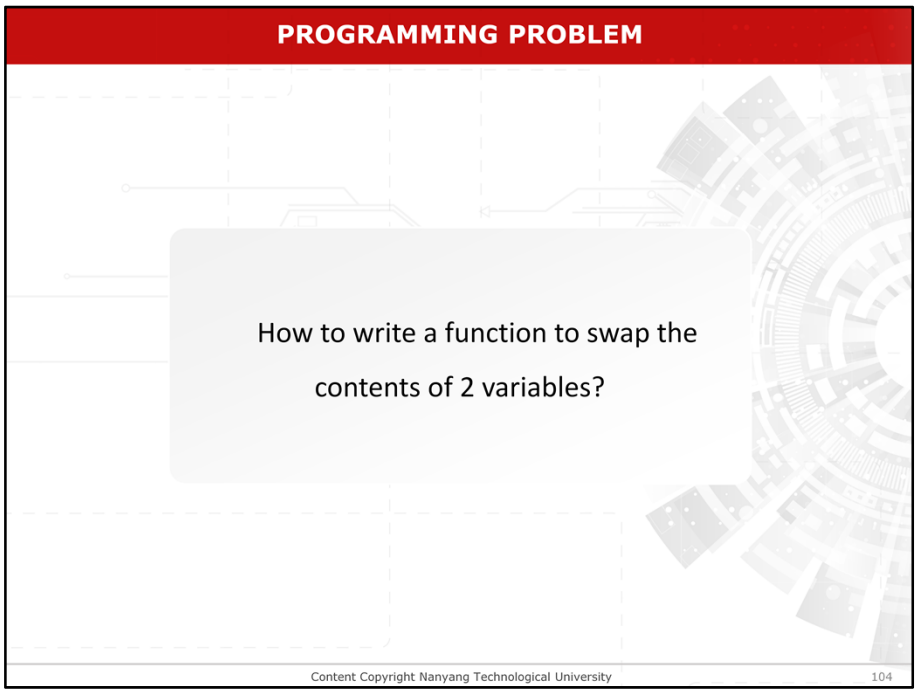



Which in turn store the address of a variable (in this case **a**) of the corresponding primitive type.

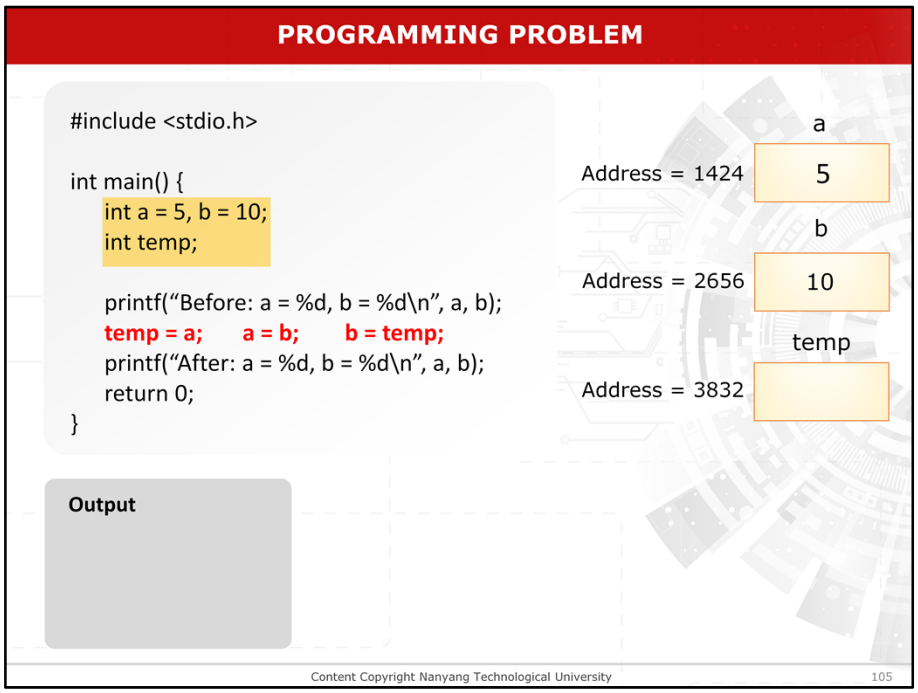



Therefore, **pp will have the value stored in a, that is, 3.

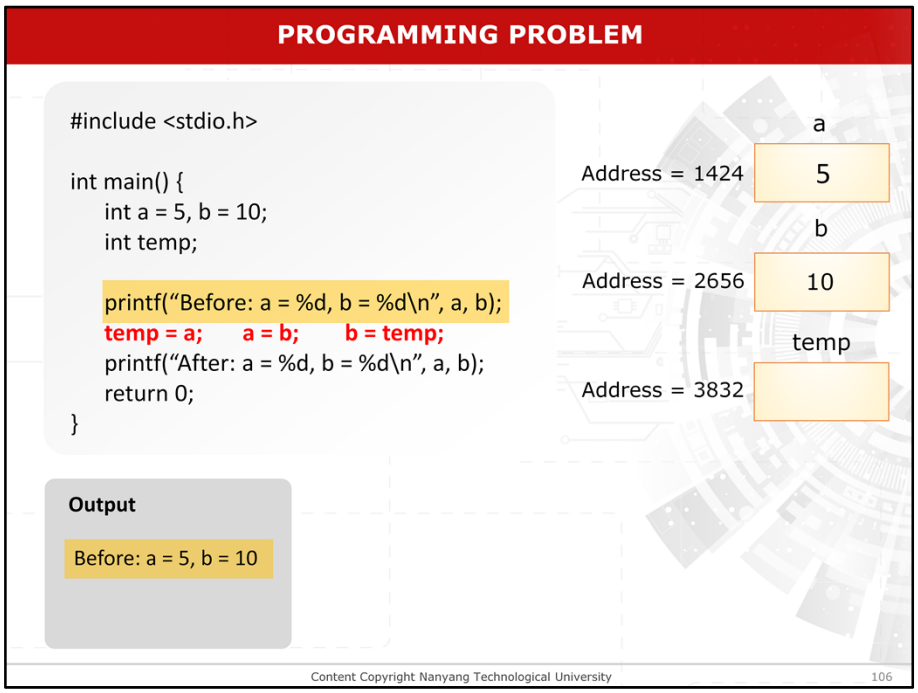
This example shows the use of double indirection.



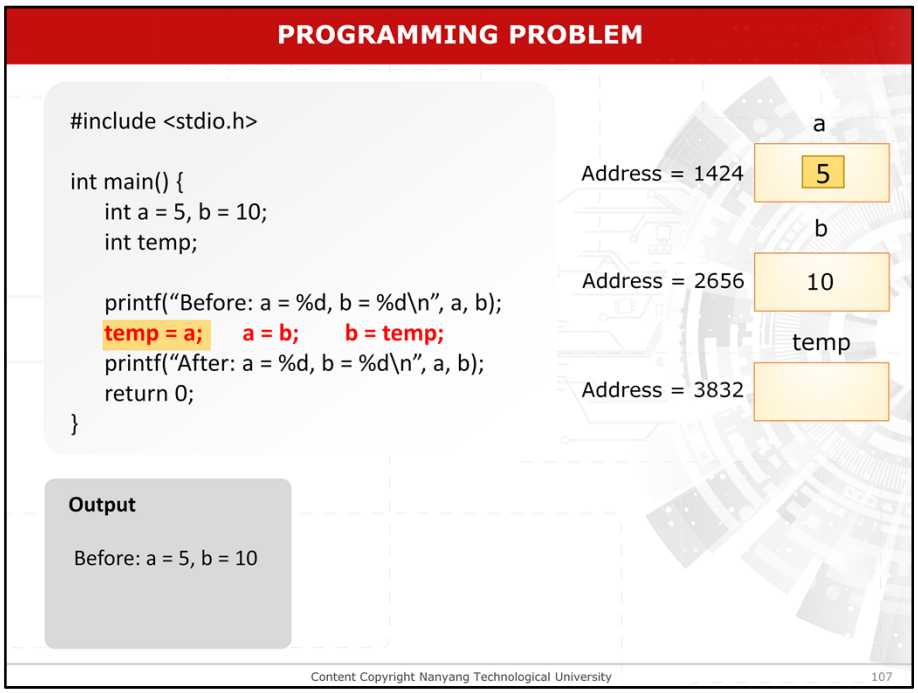

A programing problem given is to write a function that can swap the values of two parameters of the function.




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

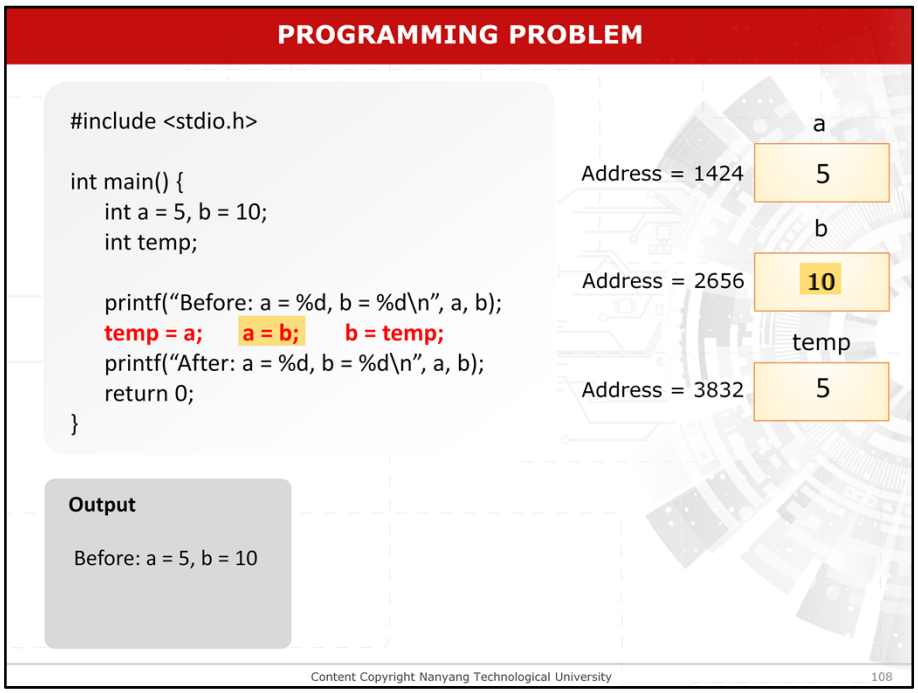



Executing the printf statement will give the output as shown.



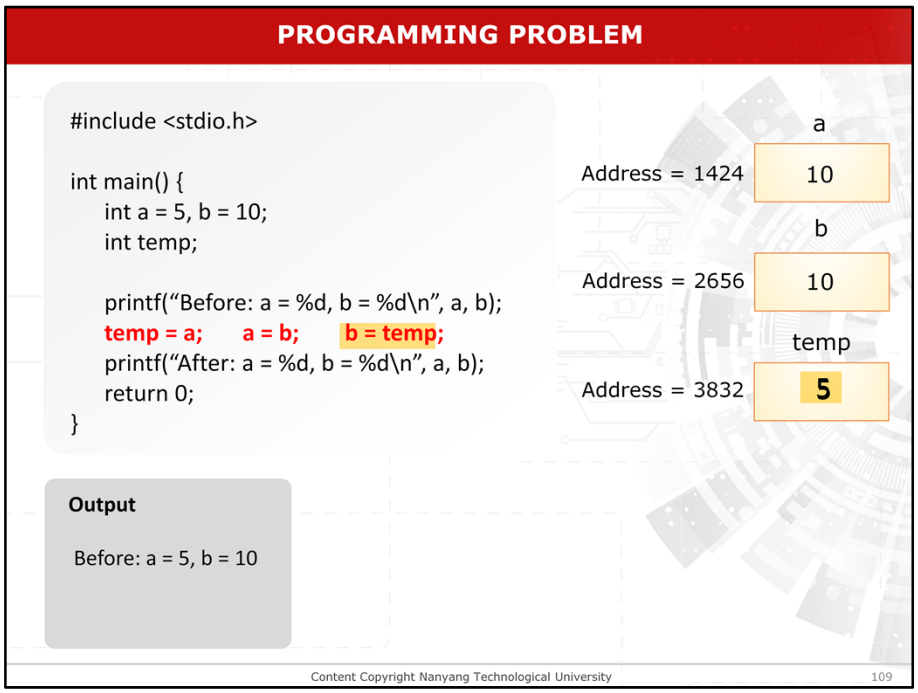

Variable temp has been assigned the value of a

*[+animations]*




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

*[+animations]*

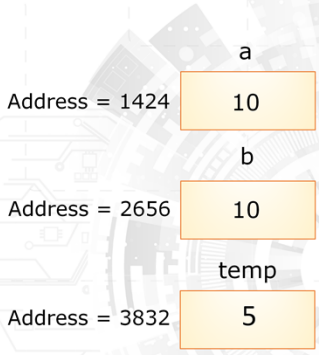



Then the value of b is reassigned to be value of temp. So the value of b change from 10 to 5.

*[+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;
}
```



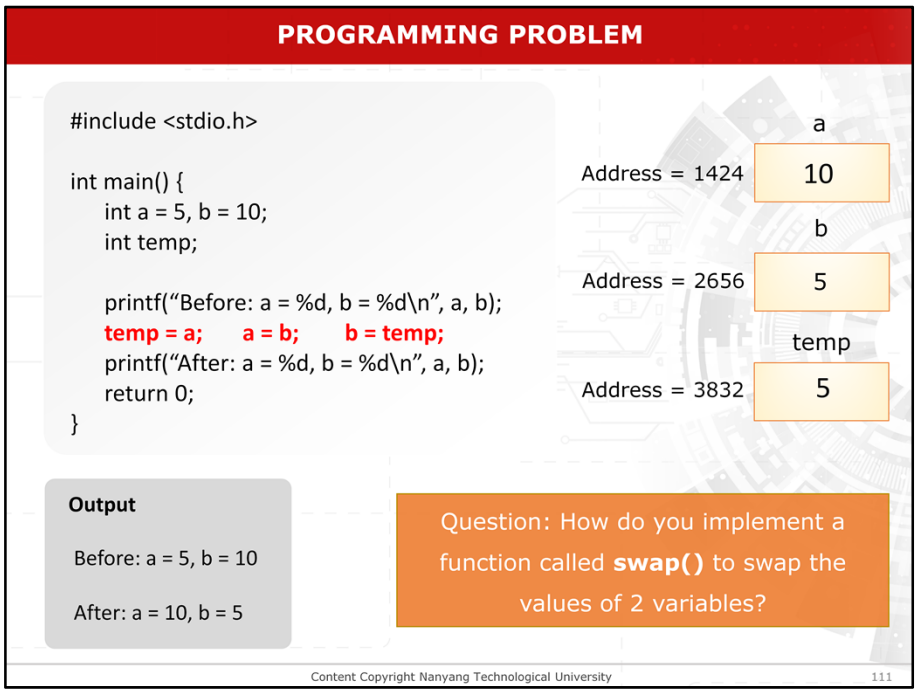

**Output**

Before: a = 5, b = 10

After: a = 10, b = 5




Thus, the output shows the swap required.




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

## PROGRAMMING PROBLEM

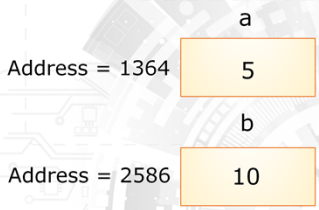
```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;

    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 */
 }
```




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 */
 }
```



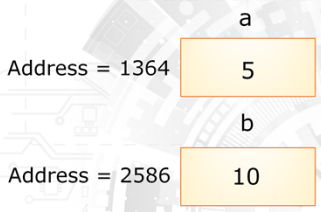




In the **main()** function of the program template, it declares two variables **a** and **b**.

## PROGRAMMING PROBLEM

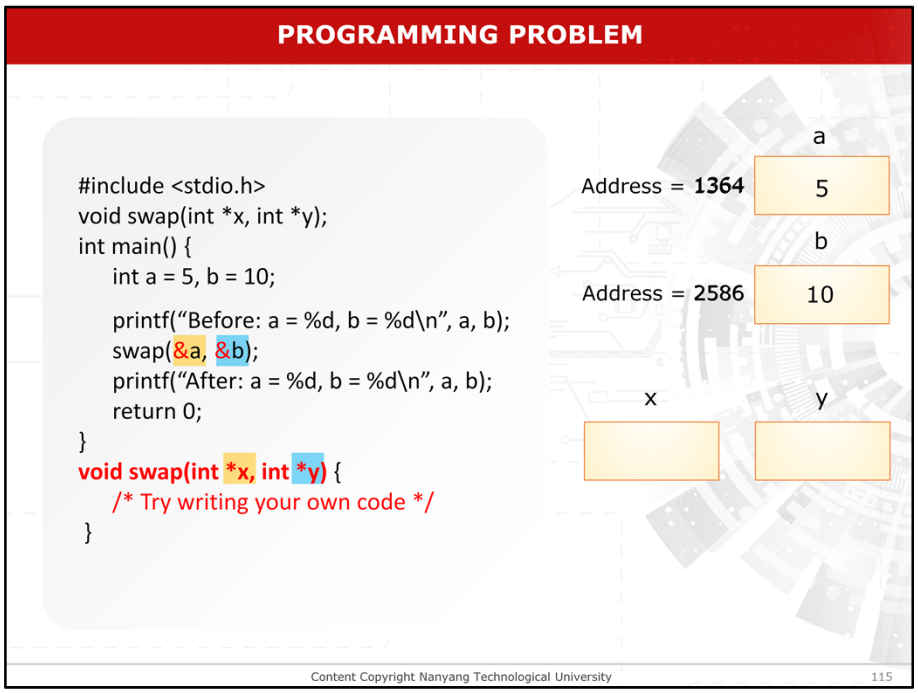
```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;

    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 */
 }
```







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

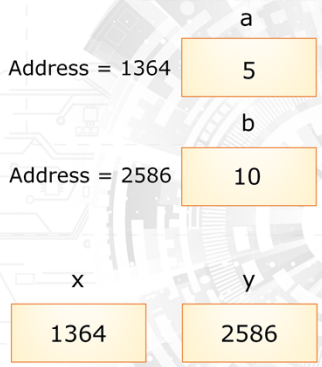



the addresses of the two variables are passed into the calling function.

*[+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 */
 }
```



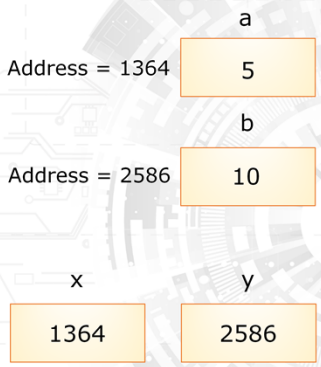




In the function **swap()**, it swaps the contents of the two parameters **x** and **y**, which contain the addresses of the variables **a** and **b** passed in from the **main()** function. It is done via **call by reference**, as the function is required to return two values back to the calling function.

## 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 */
 }
```



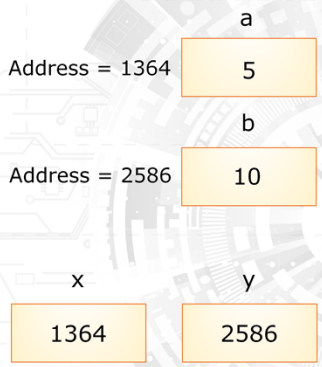




Question: How do you implement the function **swap()**?

## PROGRAMMING PROBLEM

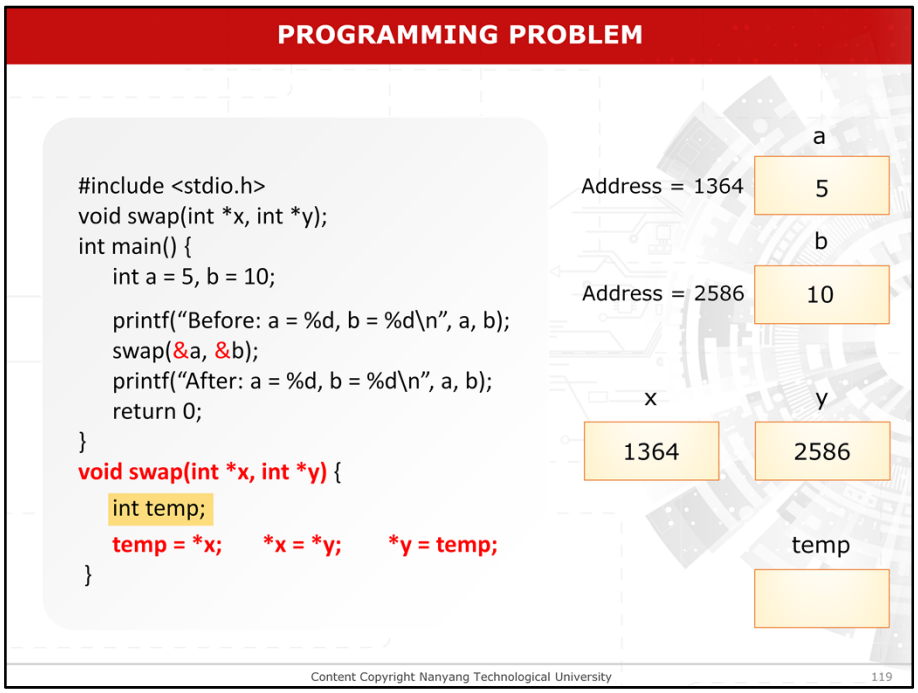
```
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 5, b = 10;
    printf("Before: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After: a = %d, b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;      *x = *y;      *y = temp;
}
```



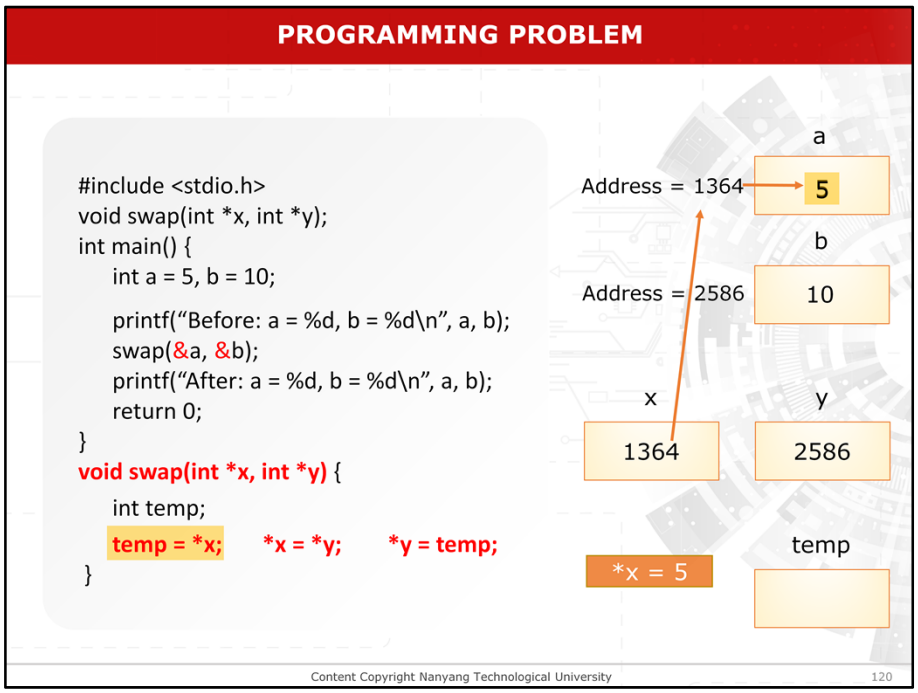




In the **swap()** function, a simple swap operation is performed based on the parameters **x** and **y** via call by reference.

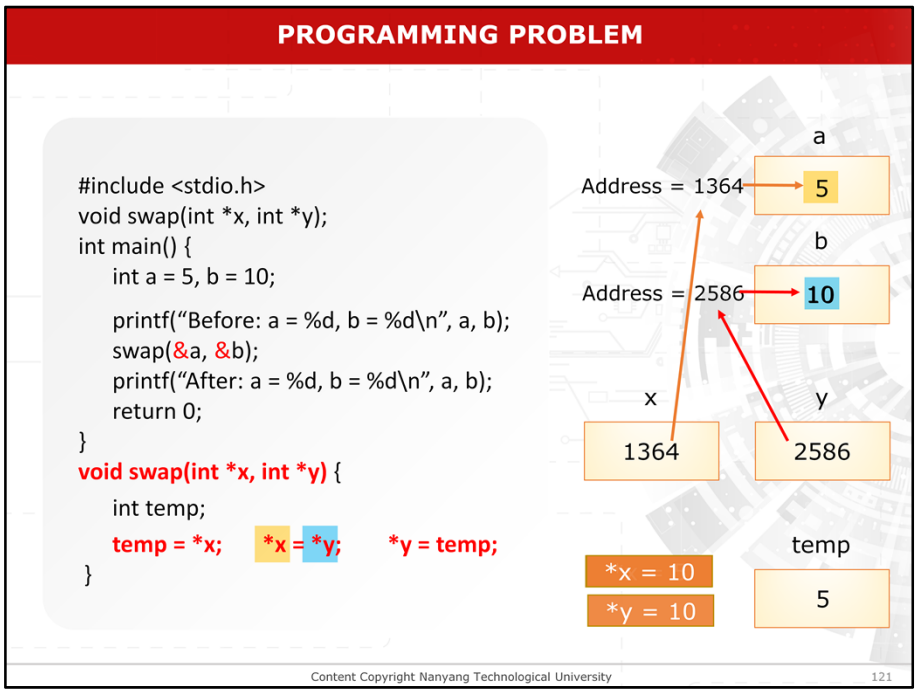



A parameter temp is declared.



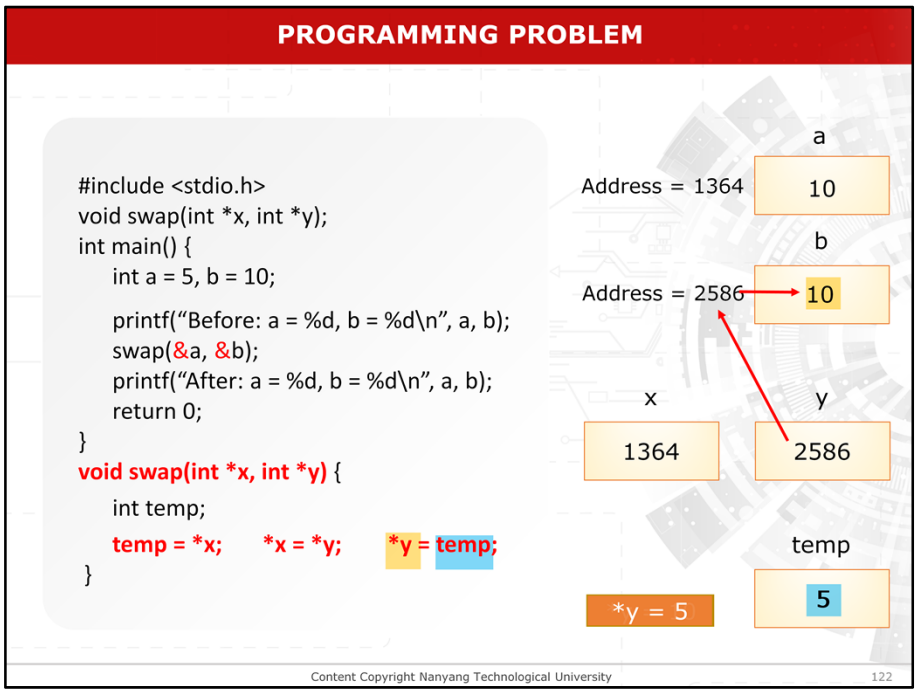

The value of temp contains the value stored in parameter a.

*[+animations]*



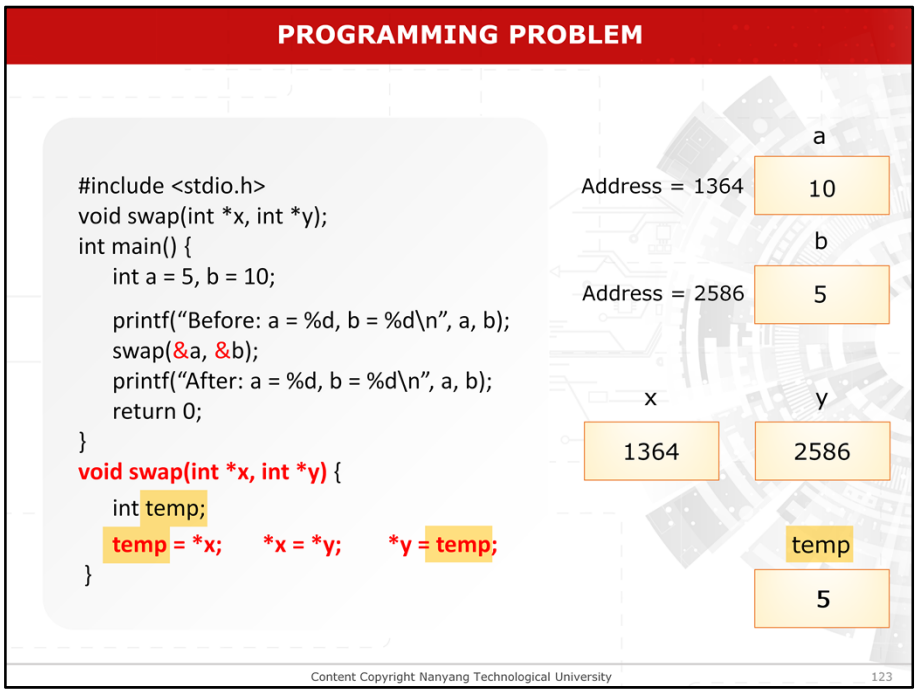

The value of a changes from 5 to 10.

*[+animations]*

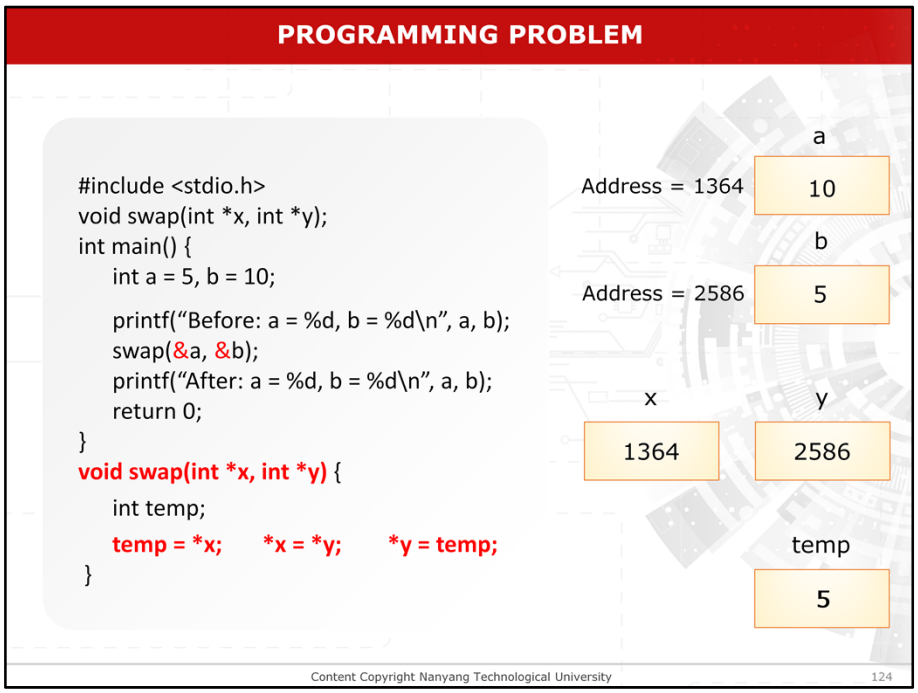



The value of b changes from 10 to 5.

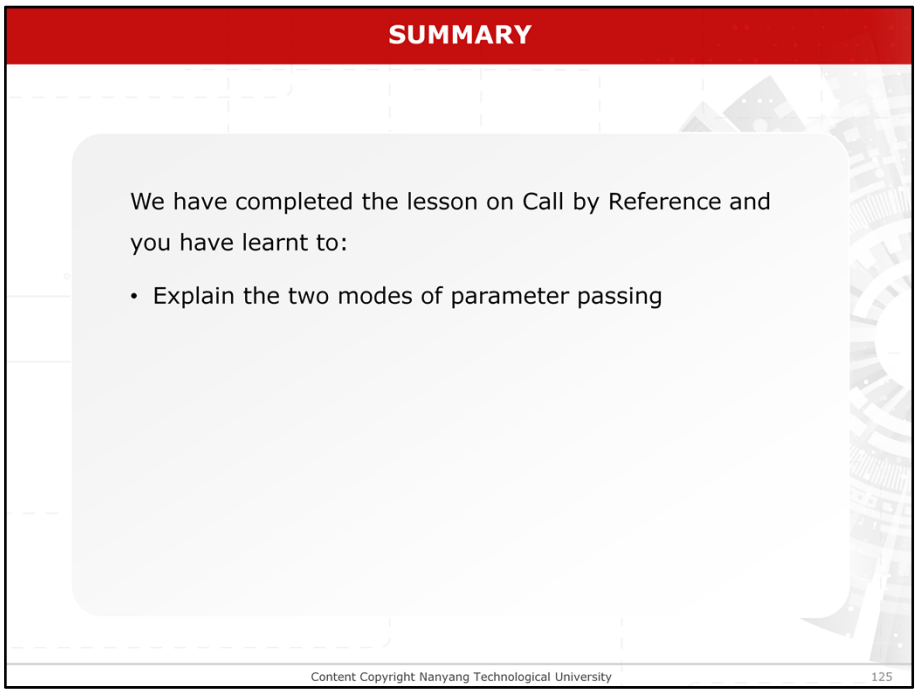
*[+animations]*




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

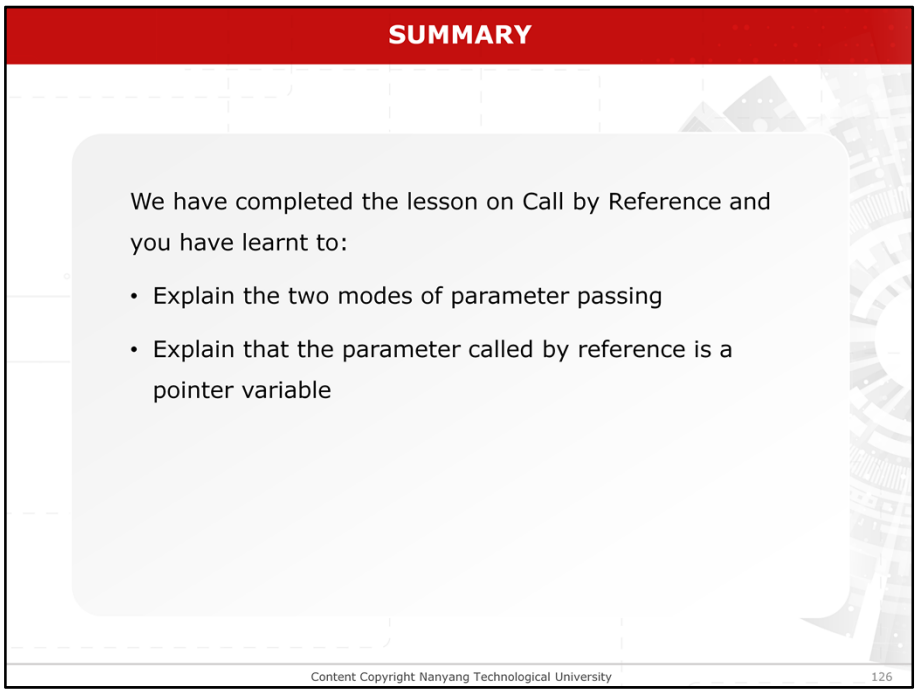



This example shows us how to use pointer dereferencing for the swapping operation.

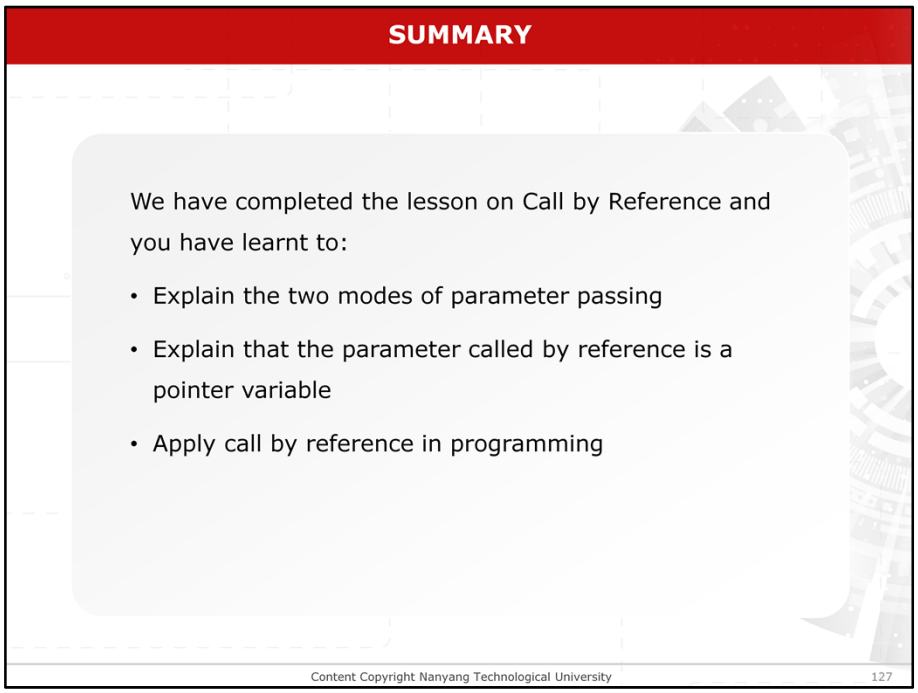



We have completed the lesson on Call by Reference and you have learnt to:

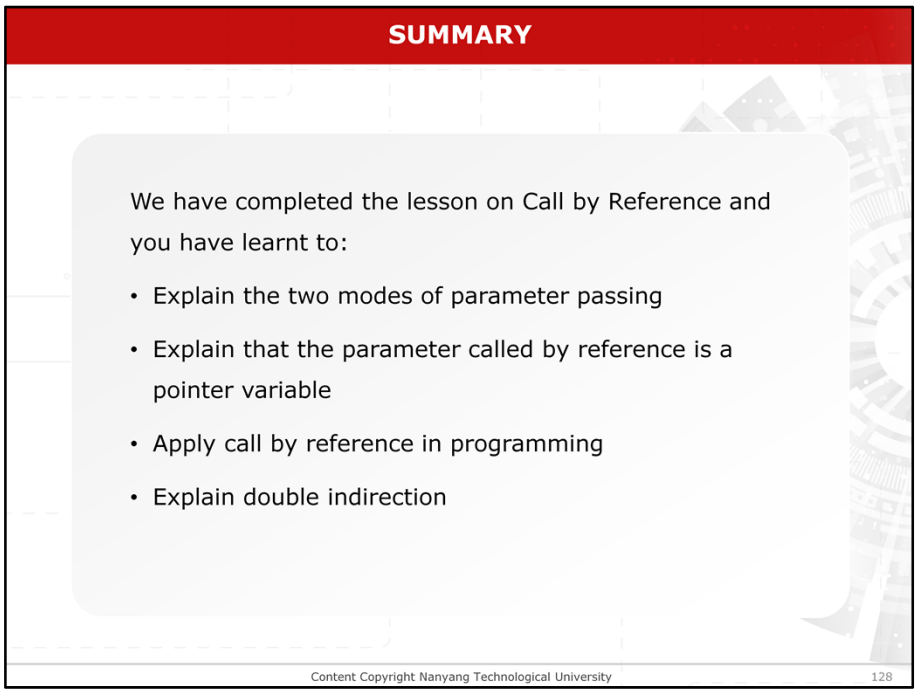
Explain the two modes of parameter passing



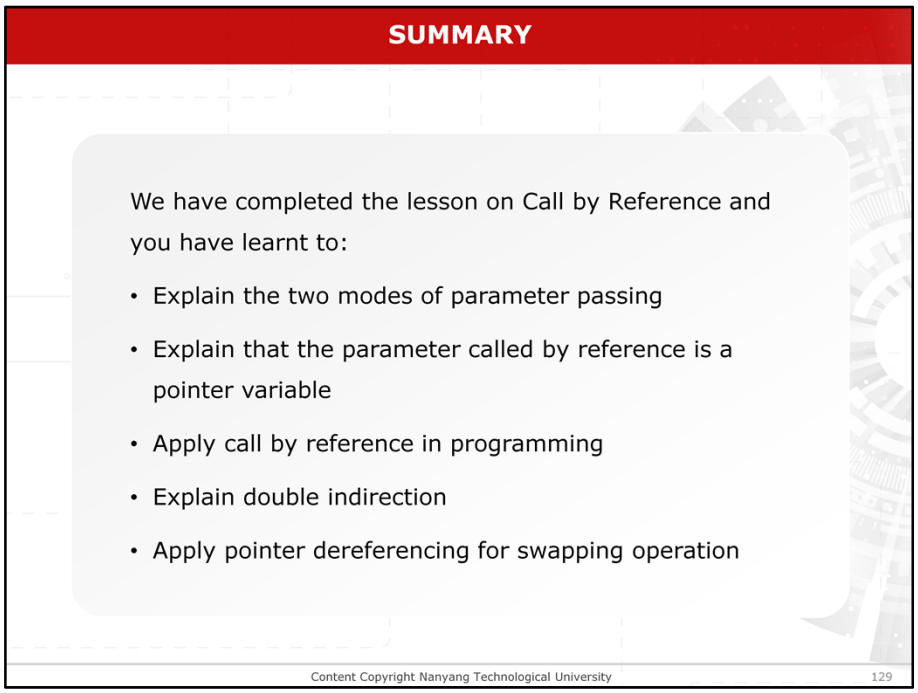

Explain that the parameter called by reference is a pointer variable




Apply call by reference in programming




Explain double indirection




And apply pointer dereferencing for swapping operation.