

Lesson 6.5 Arrays as Function Arguments.

In this lesson, we will learn about pointer constants.

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};
```

Content Copyright Nanyang Technological University

2

For example, the array table has been declared.

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};  
maximum(table, 10);
```

Content Copyright Nanyang Technological University

3

Assume a function called **maximum()** that finds the maximum value of the array table.

We may call the function as follows:

```
maximum(table, 10);
```

To receive the array argument, the function must be defined with a parameter that is an array. There are three ways to define a function with a one-dimensional array as the argument.

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};  
maximum(table, 10);  
  
void maximum(int table[ ], int n) // n is the data size  
{  
    ....  
}
```

Content Copyright Nanyang Technological University

4

The first way is to define the function as

integer maximum (integer table[], integer n)

The parameter list includes an array **table** and an integer **n**. The data type of the array is specified, and empty square brackets follow the array name. The integer **n** is used to indicate the size of the array.

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};
```

```
maximum(table, 10);
```

```
void maximum(int table[ ], int n)
{
    ....
}
```

// n is the data size

```
void maximum(int table[TABLESIZE])
{
    ....
}
```

Content Copyright Nanyang Technological University

5

Another way is to define the function as

integer maximum(integer table [TABLESIZE])

The parameter list includes an array only. The array size **TABLESIZE** is also specified in the square brackets of the array **table**.

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};  
maximum(table, 10);  
  
void maximum(int table[ ], int n)      // n is the data size  
{  
    ....  
}  
  
void maximum(int table[TABLESIZE])  
{  
    ....  
}  
  
void maximum(int *table, int n)  
{  
    ....  
}
```

Content Copyright Nanyang Technological University

6

The third way is to define the function as

integer maximum(integer asterisk table, integer n)

The parameter list includes a pointer **table** of type **integer**, and an integer **n**. The integer **n** is used to indicate the size of the array.

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};
```

```
maximum(table, 10);
```

```
void maximum(int table[ ], int n)
{
    ....
}
```

```
void maximum(int table[ ], int n);
```

```
void maximum(int table[TABLESIZE])
{
    ....
}
```

```
void maximum(int *table, int n)
{
    ....
}
```

Content Copyright Nanyang Technological University

7

The function prototypes of the function become
integer maximum (integer table[], integer n);

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};
```

```
maximum(table, 10);
```

```
void maximum(int table[ ], int n)
{
    ....
}
```

```
void maximum(int table[TABLESIZE])
{
    ....
}
```

```
void maximum(int *table, int n)
{
    ....
}
```

```
void maximum(int table[ ], int n);
```

```
void maximum(int table[TABLESIZE]);
```

Content Copyright Nanyang Technological University

8

or integer maximum (integer table [TABLESIZE]);

ARRAYS AS FUNCTION ARGUMENTS: FUNCTION HEADER

```
int table[10] = {34,21,65,54,17,48,29,93,49,23};
```

```
maximum(table, 10);
```

```
void maximum(int table[ ], int n)
{
    ....
}
```

```
void maximum(int table[TABLESIZE])
{
    ....
}
```

```
void maximum(int *table, int n)
{
    ....
}
```

```
void maximum(int table[ ], int n);
```

```
void maximum(int table[TABLESIZE]);
```

```
void maximum(int *table, int n);
```

Content Copyright Nanyang Technological University

9

or integer maximum (integer asterisk table, integer n);

ARRAYS AS FUNCTION ARGUMENTS: CALLING THE FUNCTION

Any dimensional array can be passed as a function argument, e.g. we can **call the function**:

```
fn(table, n); /* calling a function */
```

where **fn()** is a function and **table** is a 1-D array, and **n** is the size of the array **table**.

Content Copyright Nanyang Technological University

10

An array can also be passed to a function as an argument, e.g.,

f n (table, n);

where **f n** is a function and **table** is a 1-D array.

When we pass an array as a function argument, the original array is passed to the function. There is no local copy of the array to be maintained in the function. This is mainly due to efficiency as arrays can be quite large and thereby taking a considerably large storage space if a local copy is stored.

ARRAYS AS FUNCTION ARGUMENTS: CALLING THE FUNCTION

Any dimensional array can be passed as a function argument, e.g. we can **call the function**:

```
fn(table, n);           /* calling a function */
```

where **fn()** is a function and **table** is a 1-D array, and **n** is the size of the array **table**.

An **array table** is passed in using **call by reference** to a function.

Content Copyright Nanyang Technological University

11

In fact, the array is passed using *call by reference* to the function.

ARRAYS AS FUNCTION ARGUMENTS: CALLING THE FUNCTION

Any dimensional array can be passed as a function argument, e.g. we can **call the function**:

```
fn(table, n); /* calling a function */
```

where **fn()** is a function and **table** is a 1-D array, and **n** is the size of the array **table**.

An **array table** is passed in using **call by reference** to a function. This means the **address** of the **first element** of the array is passed to the function.

Content Copyright Nanyang Technological University

12

This means that the address of the first element of the array is passed to the function. Since the function has the address of the array, any changes to the array are made to the original array.

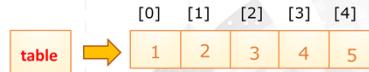
ARRAY AS A FUNCTION ARGUMENT: MAXIMUM

```
#include <stdio.h>
int maximum(int table[ ], int n);
int main()
{
    int max, index, n;
    int numArray[10];

    printf("Enter the number of values: ");
    scanf("%d", &n);
    printf("Enter %d values: ", n);
    for (index = 0; index < n; index++)
        scanf("%d", &numArray[index]);

    // find maximum
    max = maximum(numArray, n);
    printf("The maximum value is %d\n", max);

    return 0 ;
}
```



In the program, the **main()** function calls the function **maximum()** to compute the maximum value in an array.

Output

Enter the number of values:
1 2 3 4 5

The max value is 5.

Content Copyright Nanyang Technological University

13

In the program, the **main()** function calls the function **maximum()** to compute the maximum value in an array.

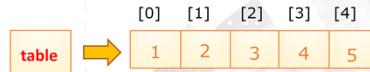
ARRAY AS A FUNCTION ARGUMENT: MAXIMUM

```
#include <stdio.h>
int maximum(int table[], int n);
int main()
{
    int max, index, n;
    int numArray[10];

    printf("Enter the number of values: ");
    scanf("%d", &n);
    printf("Enter %d values: ", n);
    for (index = 0; index < n; index++)
        scanf("%d", &numArray[index]);

    // find maximum
    max = maximum(numArray, n);
    printf("The maximum value is %d\n", max);

    return 0;
}
```



When the function **maximum()** is called, it passes an array as the function argument.

The function **maximum()** determines the maximum value stored in the array.

Output

Enter the number of values:
1 2 3 4 5

The max value is 5.

Content Copyright Nanyang Technological University

14

When the function **maximum()** is called, it passes an array as the function argument. The function **maximum()** determines the maximum value stored in the array.

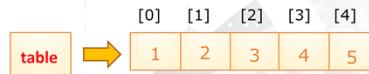
ARRAY AS A FUNCTION ARGUMENT: MAXIMUM

```
#include <stdio.h>
int maximum(int table[ ], int n);
int main()
{
    int max, index, n;
    int numArray[10];

    printf("Enter the number of values: ");
    scanf("%d", &n);
    printf("Enter %d values: ", n);
    for (index = 0; index < n; index++)
        scanf("%d", &numArray[index]);

    // find maximum
    max = maximum(numArray, n);
    printf("The maximum value is %d\n", max);

    return 0;
}
```



Apart from the array argument **numArray**, the number of elements stored in the array is also passed as an integer argument **n**.

Output

Enter the number of values:
1 2 3 4 5

The max value is 5.

Content Copyright Nanyang Technological University

15

Apart from the array argument **numArray**, the number of elements stored in the array is also passed as an integer argument **n**.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM

(1) Using index in the function implementation

Content Copyright Nanyang Technological University

16

Arrays as Function Arguments: Version 1

The implementation of the function **maximum()** uses array indexes.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM

(1) Using index in the function implementation

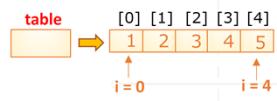
```
int maximum(int table[], int n)
{
    int i, max;
    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

Content Copyright Nanyang Technological University

17

It has two parameters: **table** and **n**.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

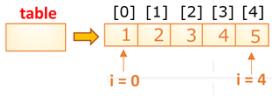
```
int maximum(int table[], int n)
{
    int i, max;
    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

Content Copyright Nanyang Technological University

18

The array is traversed element by element using indexing with **table[i]**, where **i** is the index from 0 to **n-1**, in order to find the maximum number.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

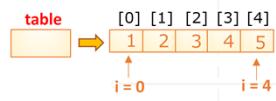
```
int maximum(int table[], int n)
{
    int i, max;
    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

Content Copyright Nanyang Technological University

19

At the end of the function, the maximum number stored in **max** is passed back to the calling function.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[], int n)
{
    int i, max;

    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

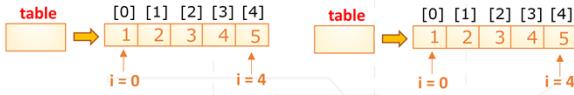
Content Copyright Nanyang Technological University

20

Arrays as Function Arguments: Version 2

The implementation of the function **maximum()** uses array base address.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[], int n)
{
    int i, max;

    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[], int n)
{
    int i, max;

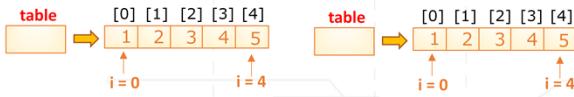
    max = *table;
    for (i = 1; i < n; i++)
        if (*(table+i) > max)
            max = *(table+i);
    return max;
}
```

Content Copyright Nanyang Technological University

21

As shown, the base address of the array **table** is used.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;

    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;

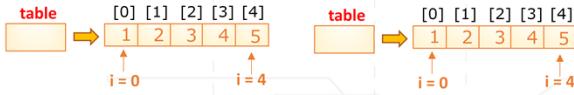
    max = *table;
    for (i = 1; i < n; i++)
        if (*(table+i) > max)
            max = *(table+i);
    return max;
}
```

Content Copyright Nanyang Technological University

22

The array element is accessed via **asterisk (table + i)**, where **i** is the index from 0 to **n-1**.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;

    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;

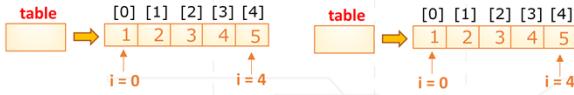
    max = *table;
    for (i = 1; i < n; i++)
        if (*(table+i) > max)
            max = *(table+i);
    return max;
}
```

Content Copyright Nanyang Technological University

23

The maximum number is then determined at the end of the loop.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;

    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;

    max = *table;
    for (i = 1; i < n; i++)
        if (*(table+i) > max)
            max = *(table+i);
    return max;
}
```

(3) Using pointer variable notation in the function implementation

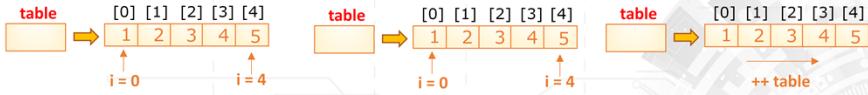
Content Copyright Nanyang Technological University

24

Arrays as Function Arguments: Version 3

The implementation of the function **maximum()** uses pointer variable notation.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;
    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;
    max = *table;
    for (i = 1; i < n; i++)
        if (*table+i) > max)
            max = *(table+i);
    return max;
}
```

(3) Using pointer variable notation in the function implementation

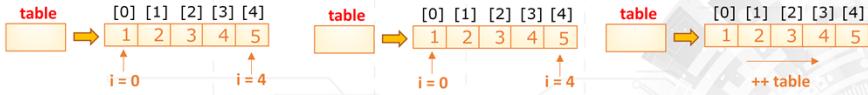
```
int maximum(int table[ ], int n)
{
    int i, max;
    max = *table;
    for (i = 0; i < n; i++)
        if (*table > max)
            max = *table;
        ++table;
    return max;
}
```

Content Copyright Nanyang Technological University

25

In this version of implementation, the array **table** is used as a pointer variable.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;
    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[ ], int n)
{
    int i, max;
    max = *table;
    for (i = 1; i < n; i++)
        if (*table+i) > max)
            max = *(table+i);
    return max;
}
```

(3) Using pointer variable notation in the function implementation

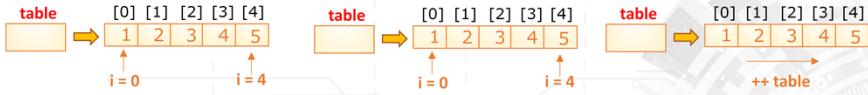
```
int maximum(int table[ ], int n)
{
    int i, max;
    max = *table;
    for (i = 0; i < n; i++) {
        if (*table > max)
            max = *table;
        ++table;
    }
    return max;
}
```

Content Copyright Nanyang Technological University

26

The array element is accessed via **asterisk table**, while it is incremented by using **table++** when traversing the array.

ARRAY AS A FUNCTION ARGUMENT: MAXIMUM



(1) Using index in the function implementation

```
int maximum(int table[], int n)
{
    int i, max;
    max = table[0];
    for (i = 1; i < n; i++)
        if (table[i] > max)
            max = table[i];
    return max;
}
```

(2) Using array base address in the function implementation

```
int maximum(int table[], int n)
{
    int i, max;
    max = *table;
    for (i = 1; i < n; i++)
        if (*table+i) > max)
            max = *(table+i);
    return max;
}
```

(3) Using pointer variable notation in the function implementation

```
int maximum(int table[], int n)
{
    int i, max;
    max = *table;
    for (i = 0; i < n; i++)
        if (*table > max)
            max = *table;
        ++table;
    return max;
}
```

Content Copyright Nanyang Technological University

27

The maximum number is then determined at the end of the loop.

QUIZ

What is the output?

```
#include <stdio.h>
int main()
{
    int a[ ] = {1,2,3,4,5,6,7,8,9,0};
    int *p = a;

    printf ("%p\n", p);
    printf ("%p\n", p+9);
    printf ("%d\n", *p+9);
    printf ("%d\n", *(p+9));
    printf ("%d\n", *++p+9);
    return 0 ;
}
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	2	3	4	5	6	7	8	9	0

0022FEE4

0022FF08

Content Copyright Nanyang Technological University

28

Quiz: Determine the output of the program.

[pause 10 sec]

QUIZ

Operator	Description	Associativity
()	Parentheses (function call)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= =	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	

Content Copyright Nanyang Technological University

29

To find the answer, you may need to understand the operator precedence table in C. Note that the precedence of dereferencing and address operators, and their relative positions when compared with the increment/decrement operators. As shown in the table, the precedence of the increment/decrement operators is higher than the dereferencing operator.

[pause 10 sec]

QUIZ

What is the output?

```
#include <stdio.h>
int main()
{
    int a[ ] = {1,2,3,4,5,6,7,8,9,0};
    int *p = a;

    printf ("%p\n", p);
    printf ("%p\n", p+9);
    printf ("%d\n", *p+9);
    printf ("%d\n", *(p+9));
    printf ("%d\n", *++p+9);
    return 0 ;
}
```



Output

```
0022FEE4 (address a[0])
0022FF08 (address a[9])
10
0
11
```

Content Copyright Nanyang Technological University

30

Did you get the correct output?
[pause 10 sec]