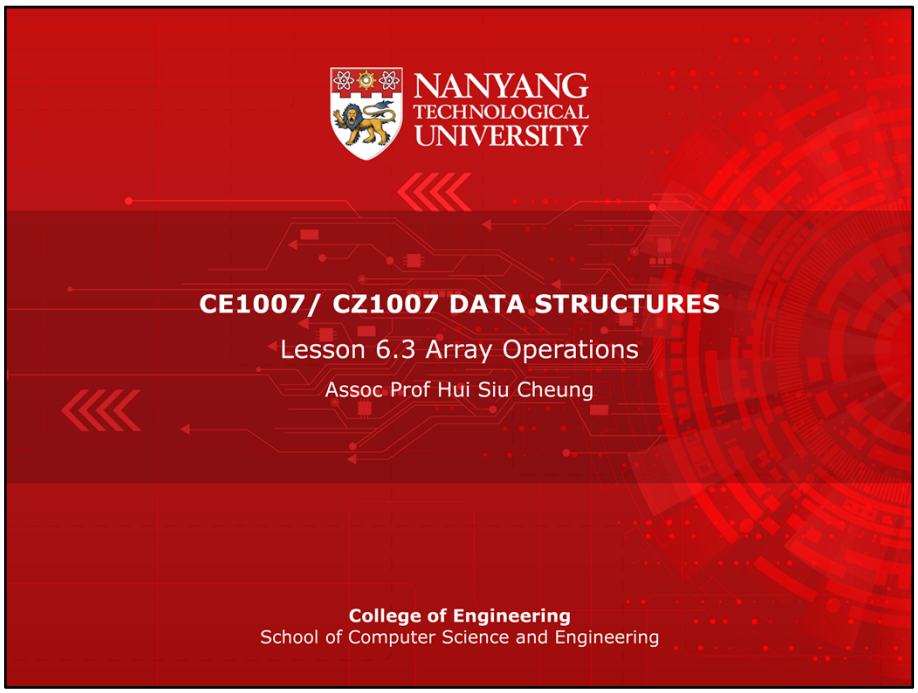
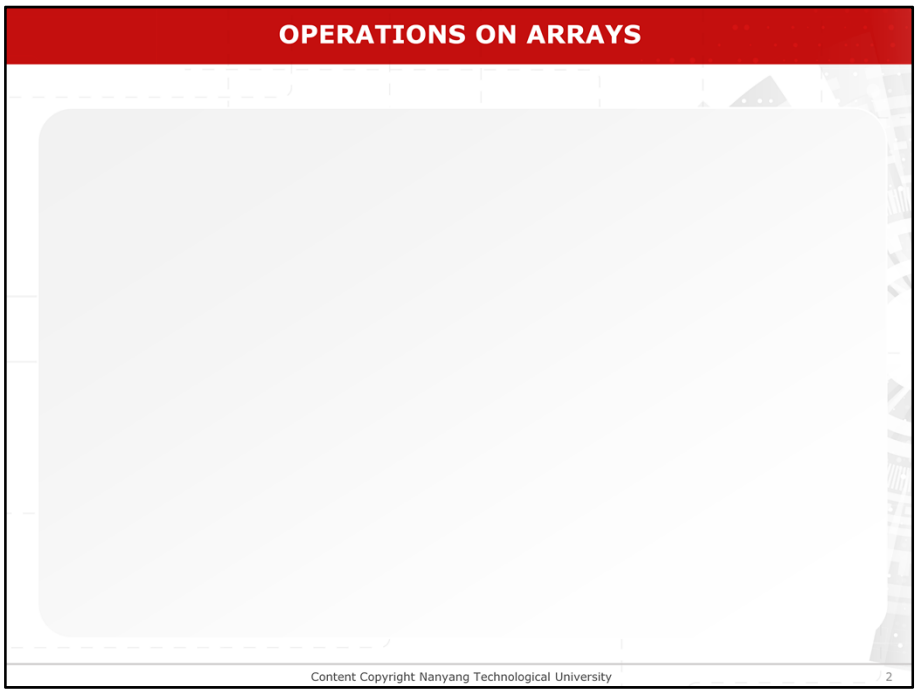


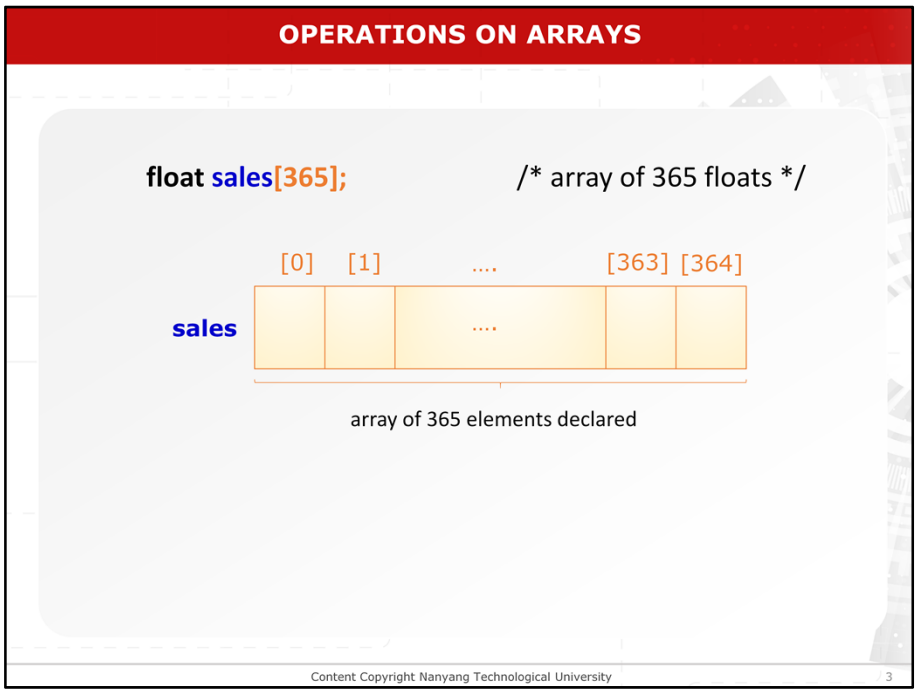
Lesson 6.3 Array Operations

In the previous video lesson, we learnt how to initialise array.

In this lesson, we will learn about the operations on arrays and a few example on traversing an array.



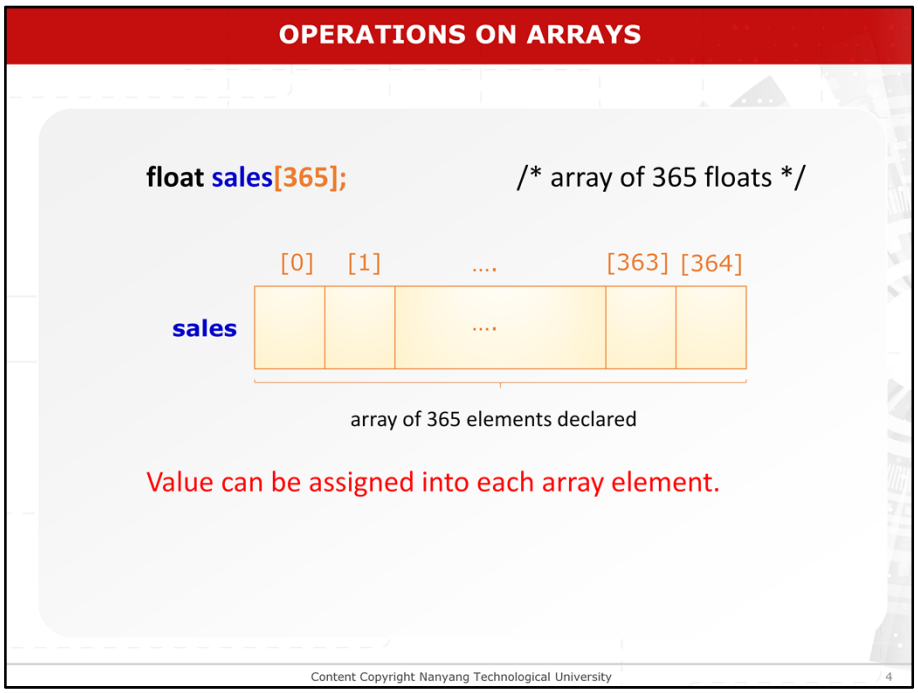
We can access array elements and perform operations on the array elements.



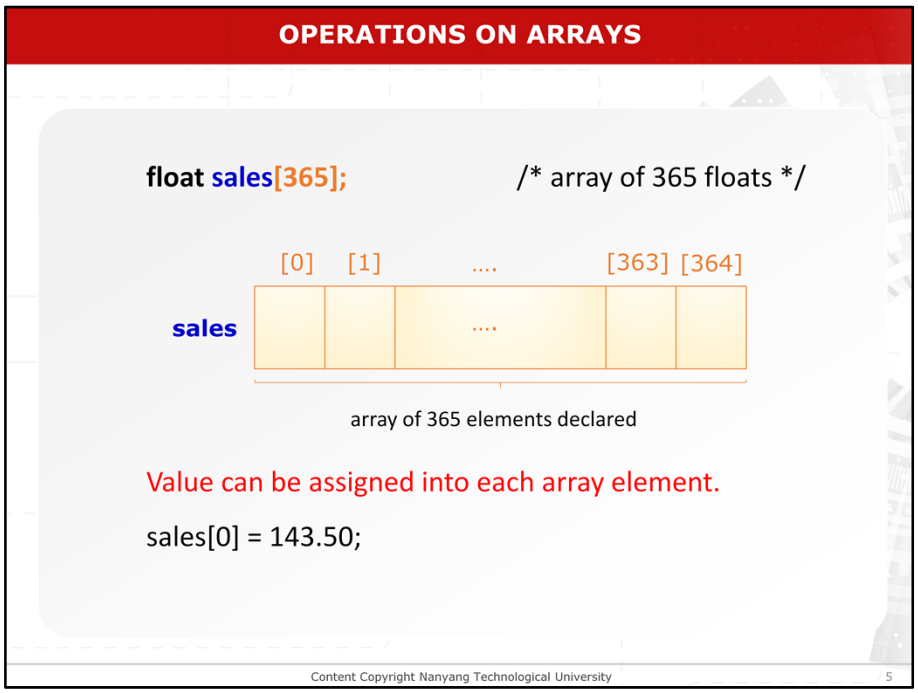
If the array variable **sales** is declared as

float sales square bracket 365 square bracket,

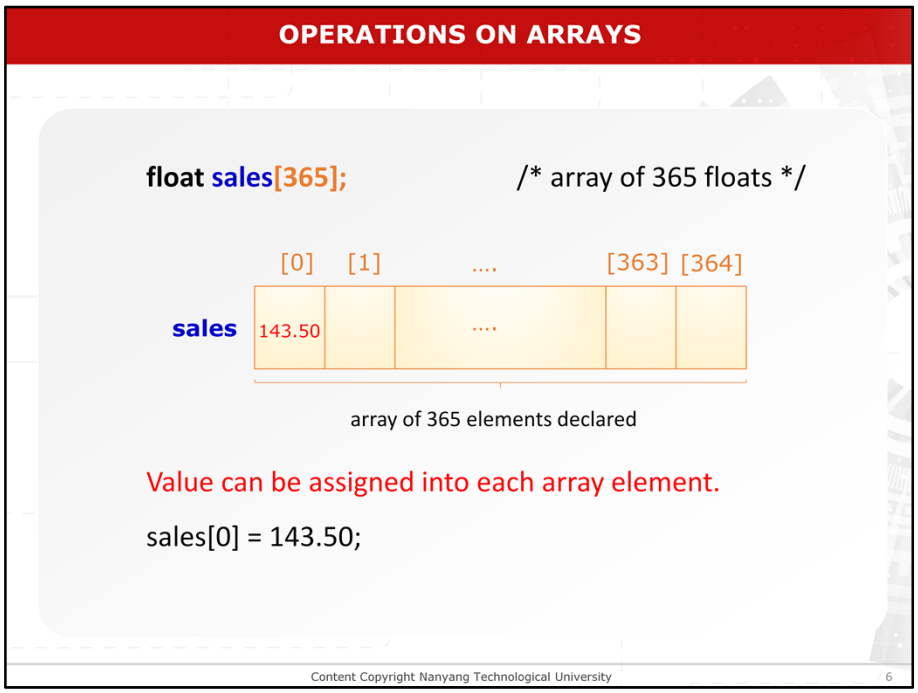
Array elements can then be processed.



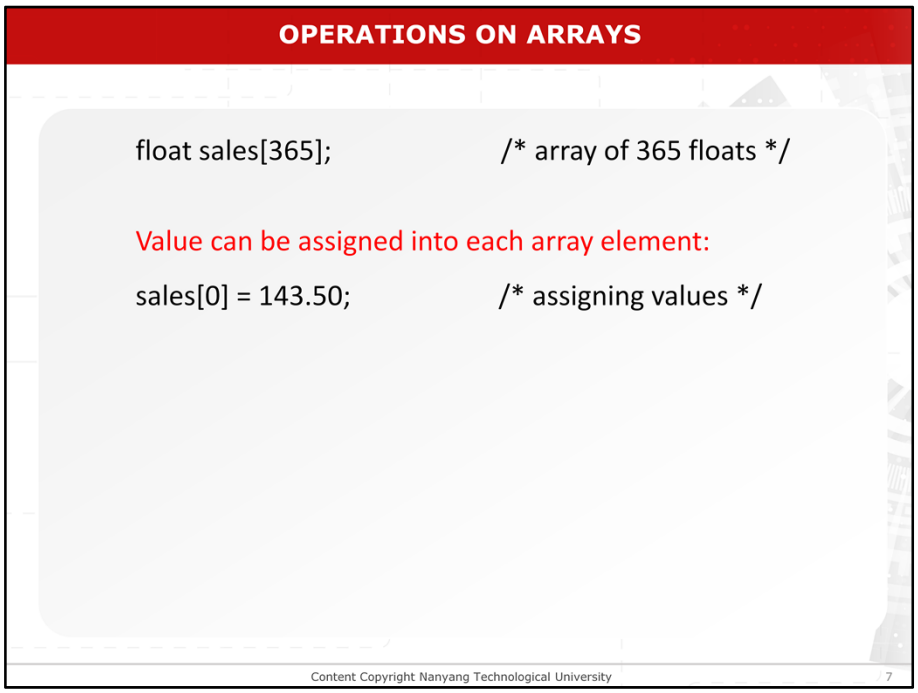
Value can be assigned into each array element.



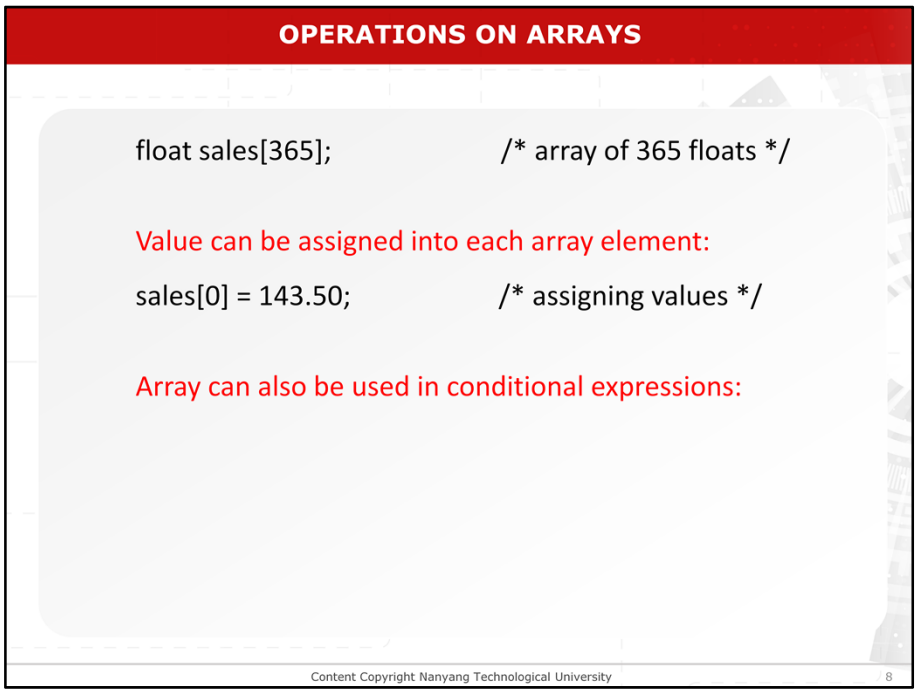
For example:
Sales square bracket 0 square bracket equals to 143.5 0;



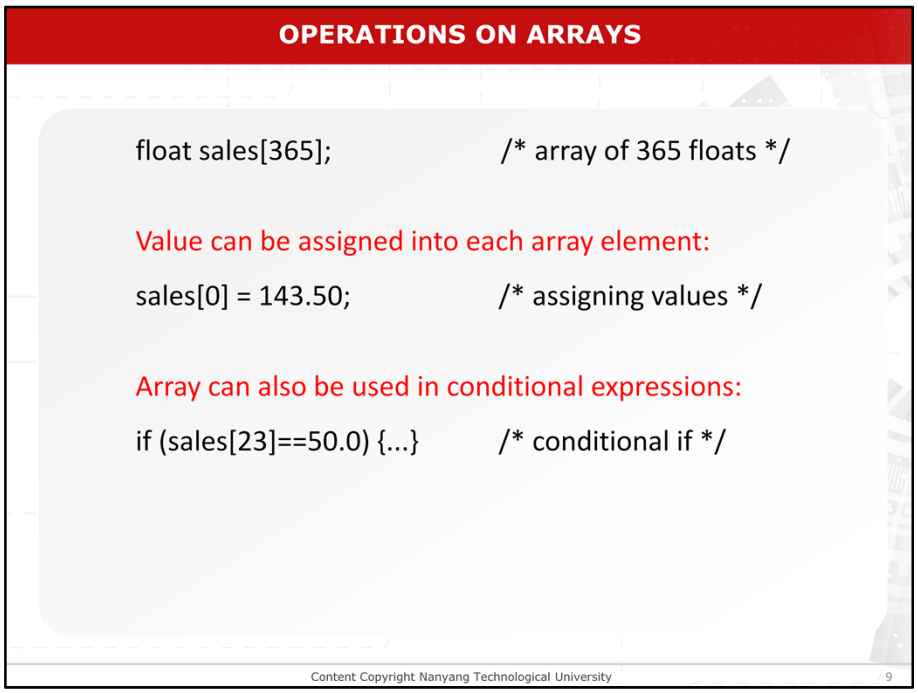
Will assign the value of 143.5 0 into the first array element.



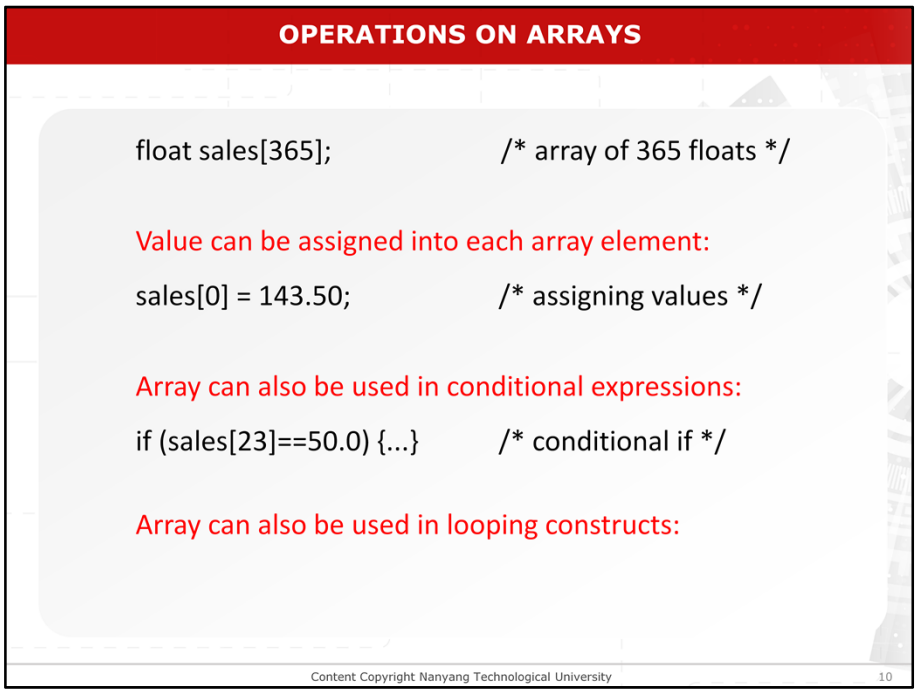
Beside assigning value into array elements,



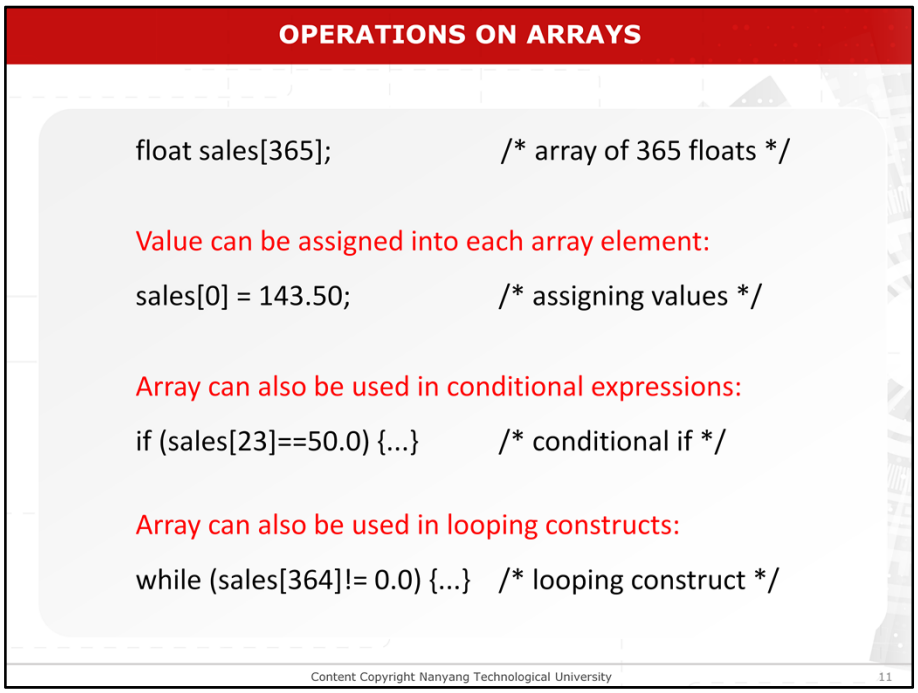
The array can also be used in conditional expressions.



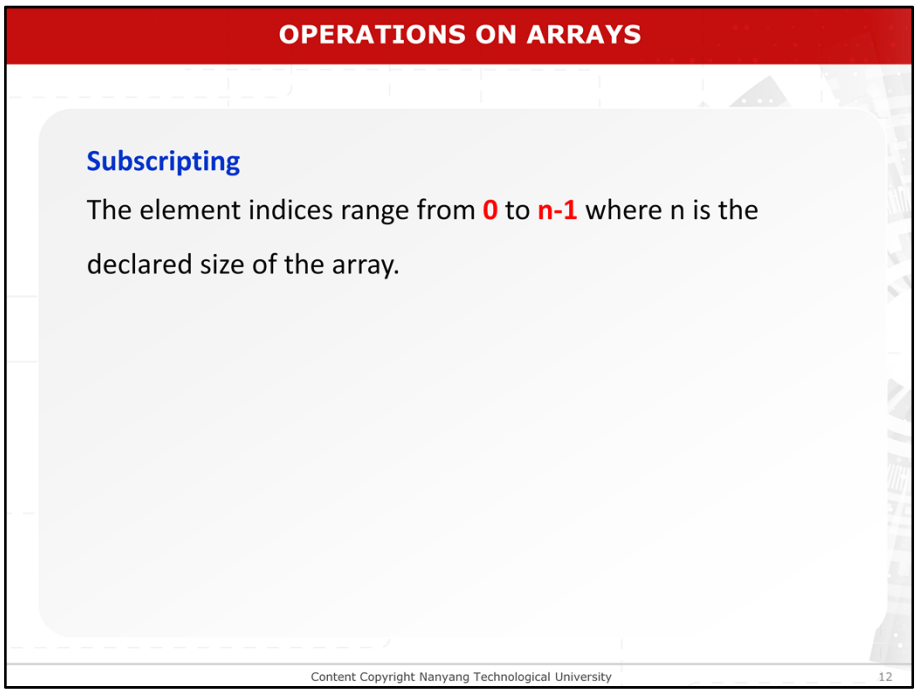
The expression shown is an example of array used in conditional expression.



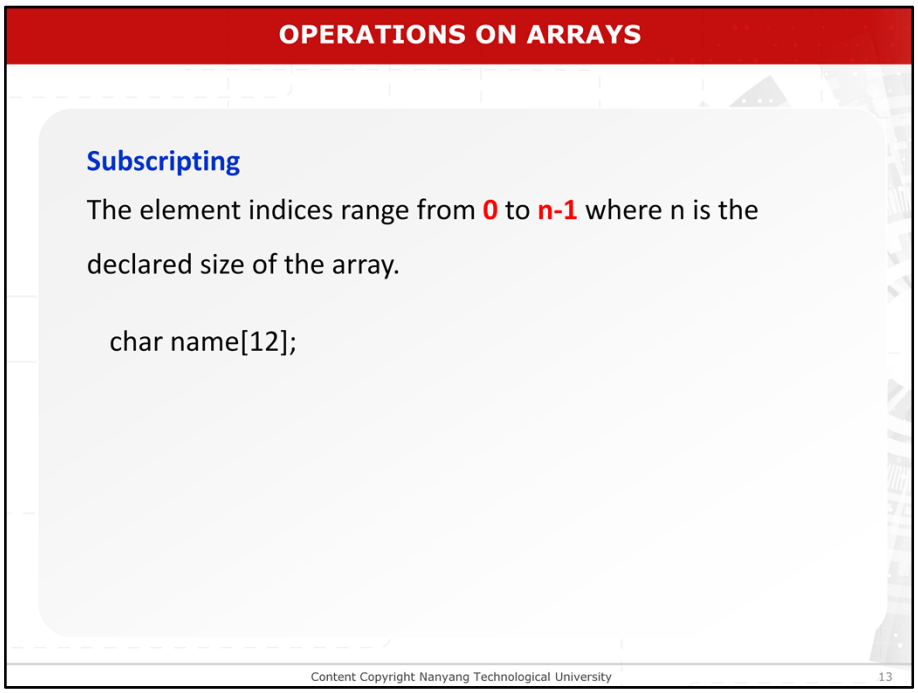
The array can also be used in looping constructs.



The expression shown is an example of array used in while loop construct.



The elements are indexed from **0** to **n minus 1** where **n** is the declared size of the array.



For the statement

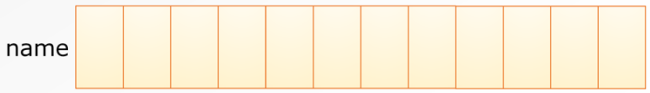
character name square bracket 12 square bracket;

OPERATIONS ON ARRAYS

Subscripting

The element indices range from **0** to **n-1** where n is the declared size of the array.

```
char name[12];
```



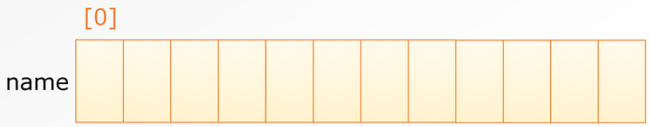
An array of size 12 elements is declared.

OPERATIONS ON ARRAYS

Subscripting

The element indices range from **0** to **n-1** where n is the declared size of the array.

```
char name[12];
```



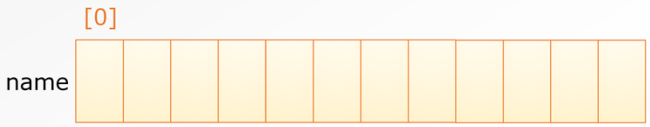
The first index is 0.

OPERATIONS ON ARRAYS

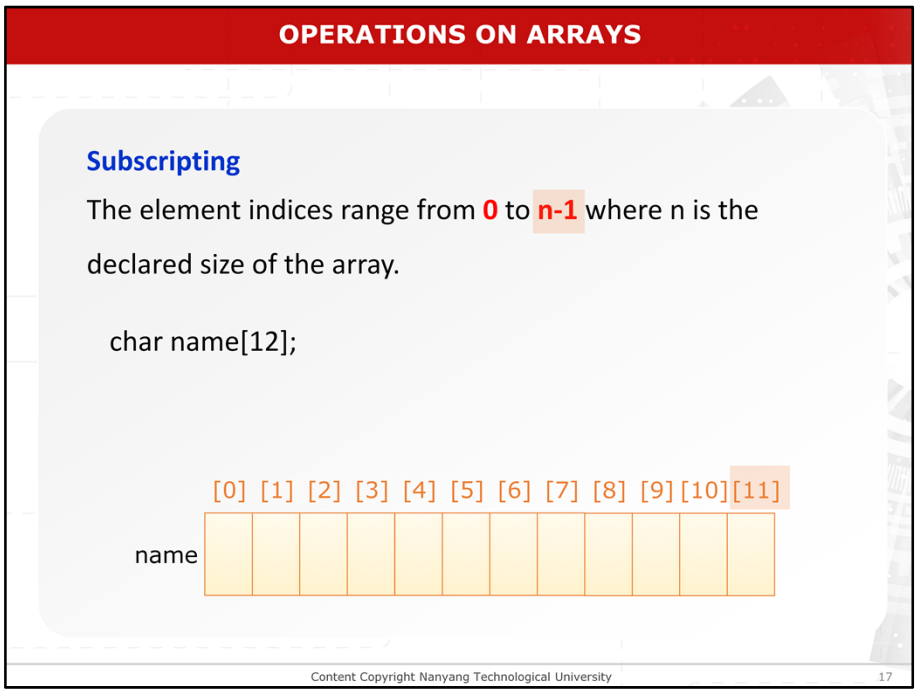
Subscripting

The element indices range from **0** to **n-1** where n is the declared size of the array.

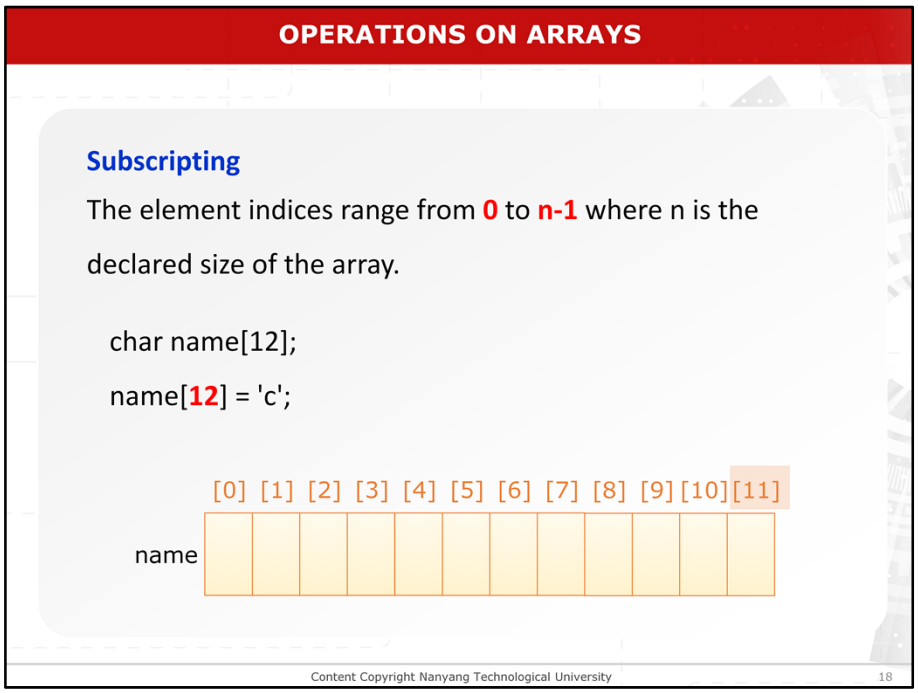
```
char name[12];
```



N is 12 for this case.



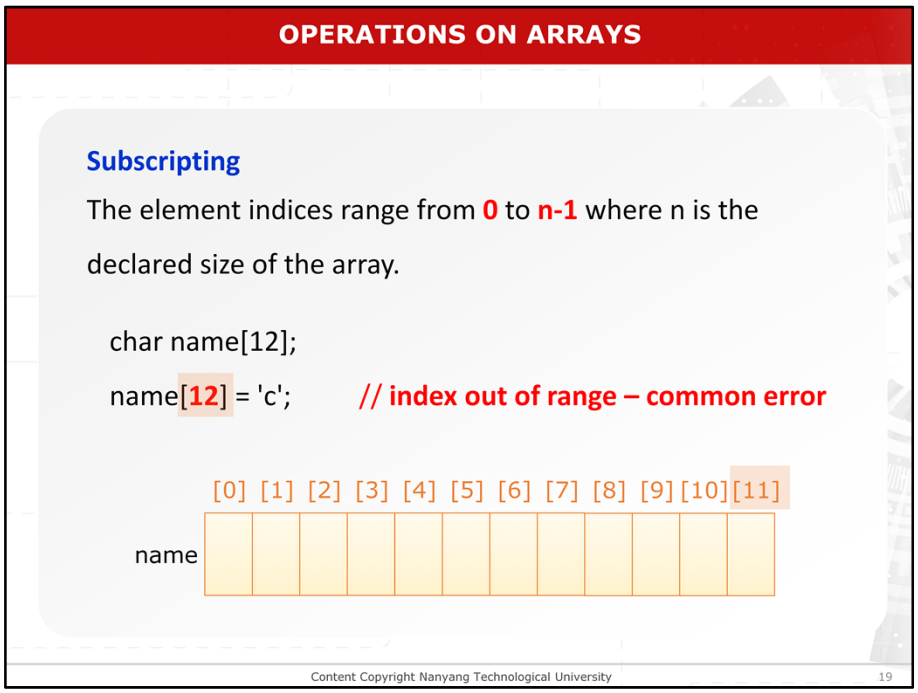
so the last index is N minus 1 that is 12 minus 1 thus 11.



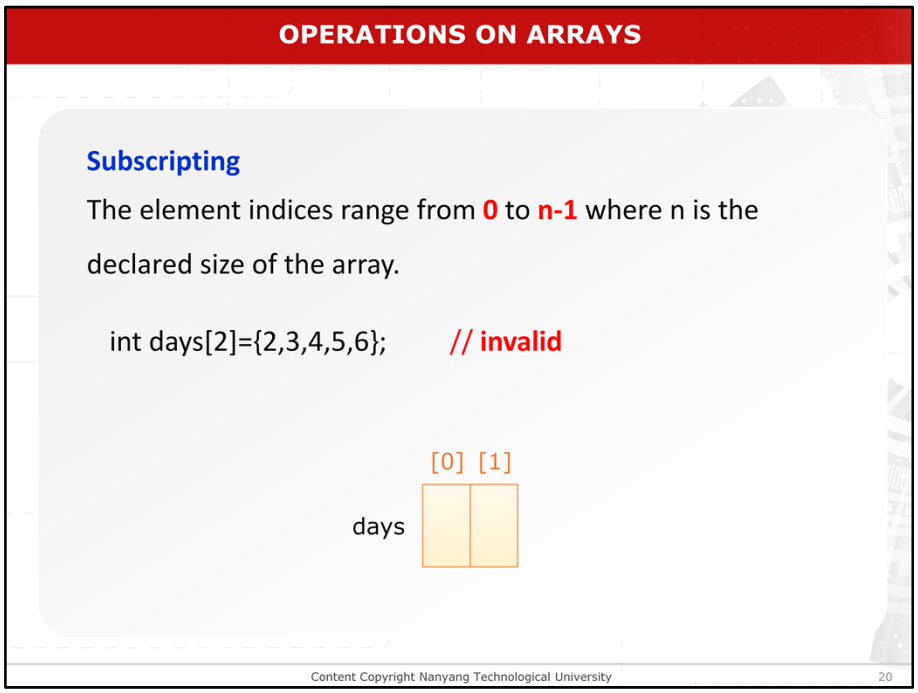
Therefore the statement

name square bracket 12 square bracket equals c

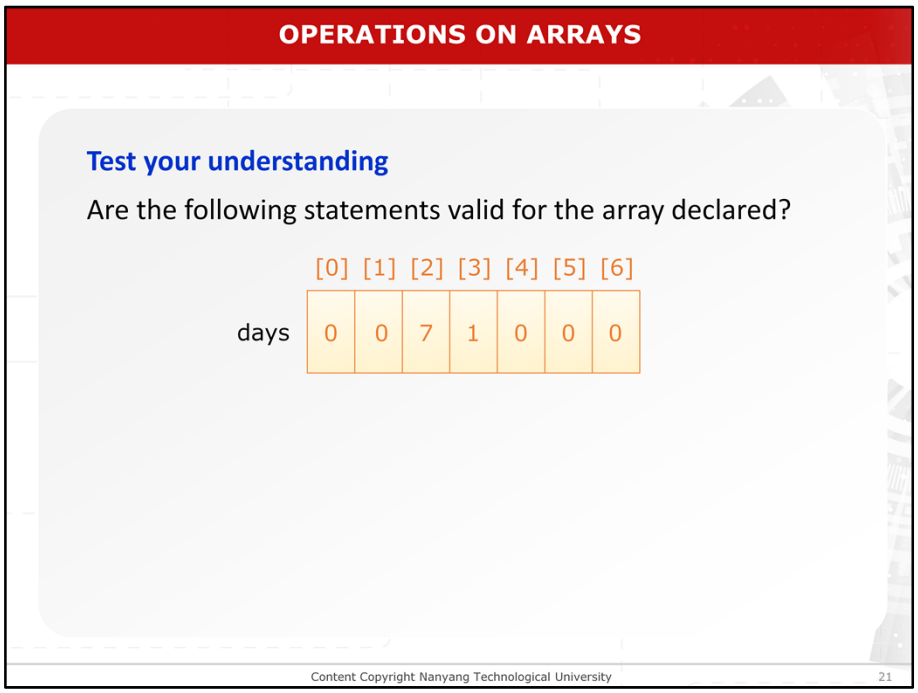
Is invalid since the array elements can only range from **name 0** to **name 11**.



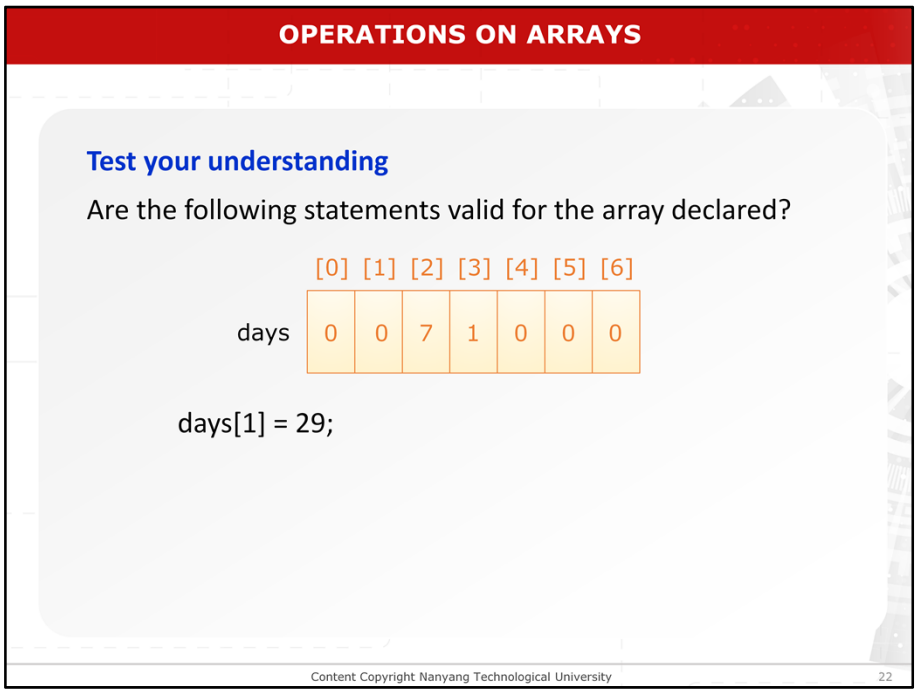
It is a common mistake to specify an index that is one value more than the largest valid index.



Similarly this statement is invalid because the array declared is of size 2 but initialized with 5 elements.

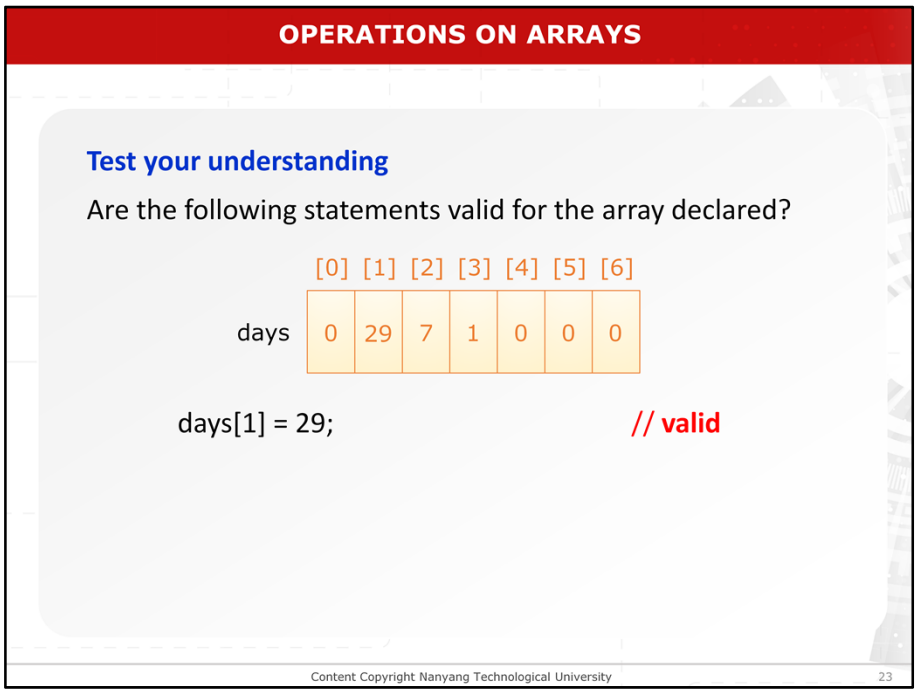


Let’s do a quick test of your understanding on what are valid or invalid statements for arrays.



Is this statement valid?

[pause for 10 sec]

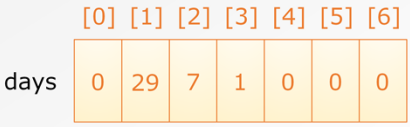


Yes. The 2nd element will be assigned the value 29.

OPERATIONS ON ARRAYS

Test your understanding

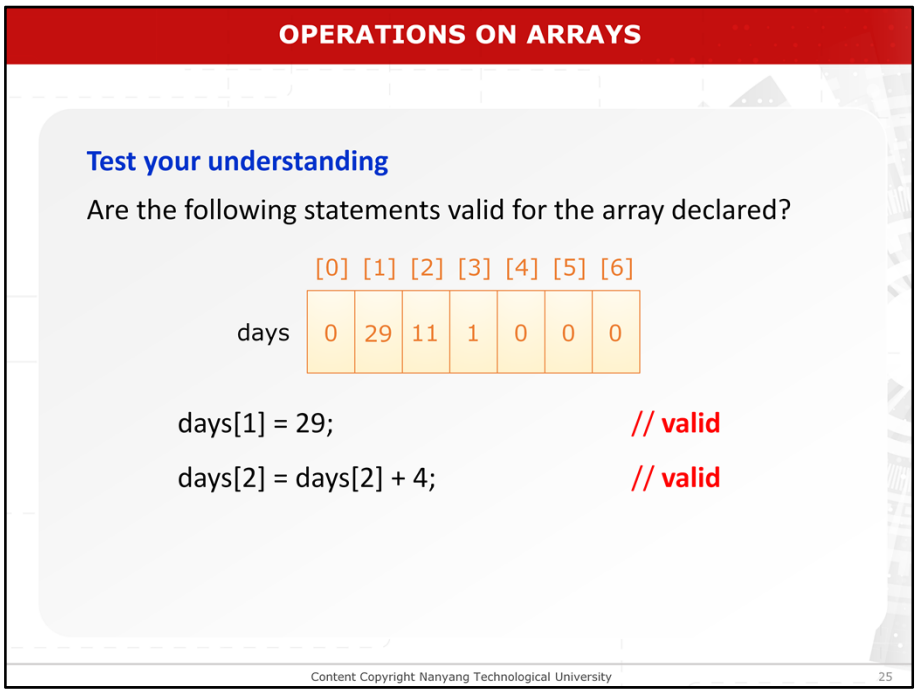
Are the following statements valid for the array declared?




```
days[1] = 29;                    // valid
days[2] = days[2] + 4;
```


Is this statement valid?

[pause for 10 sec]

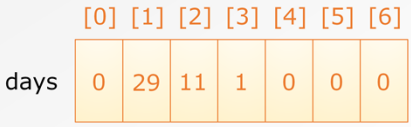


Yes. The 3rd element will be assigned the value 11 because originally it was 7 then 4 is added to it.

OPERATIONS ON ARRAYS

Test your understanding

Are the following statements valid for the array declared?




```
days[1] = 29;                          // valid
days[2] = days[2] + 4;                 // valid
days[3] = days[2] + days[3];
```

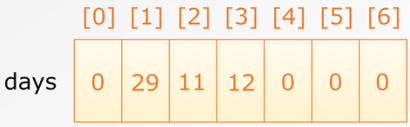

Is this statement valid?

[pause for 10 sec]

OPERATIONS ON ARRAYS

Test your understanding

Are the following statements valid for the array declared?



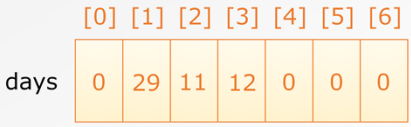

```
days[1] = 29;                          // valid
days[2] = days[2] + 4;                 // valid
days[3] = days[2] + days[3];           // valid
```


Yes. The 4th element will be assigned the value 12 because originally it was 1 then 11 is added to it.

OPERATIONS ON ARRAYS

Test your understanding

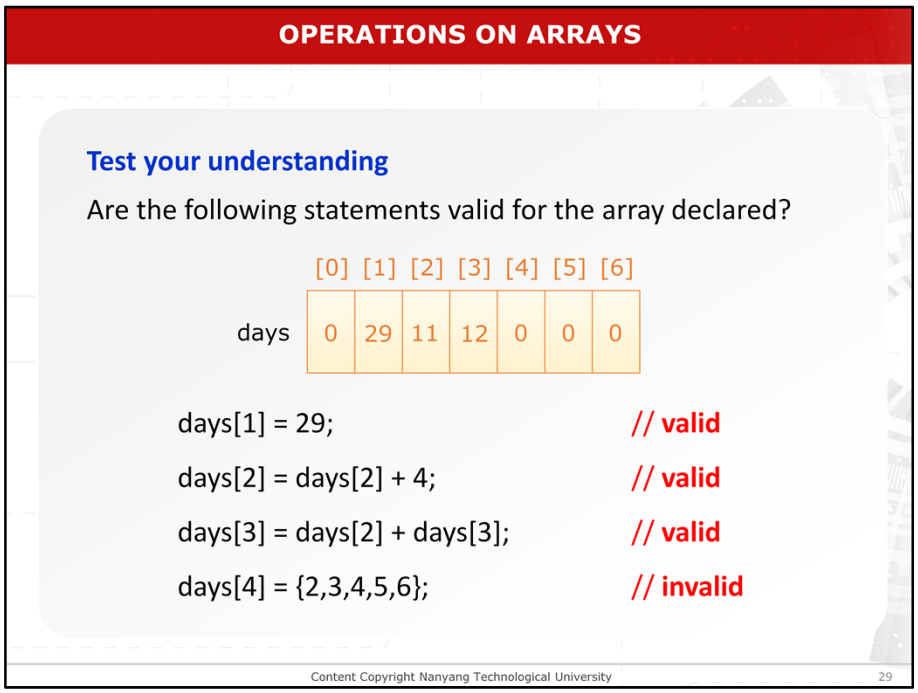
Are the following statements valid for the array declared?



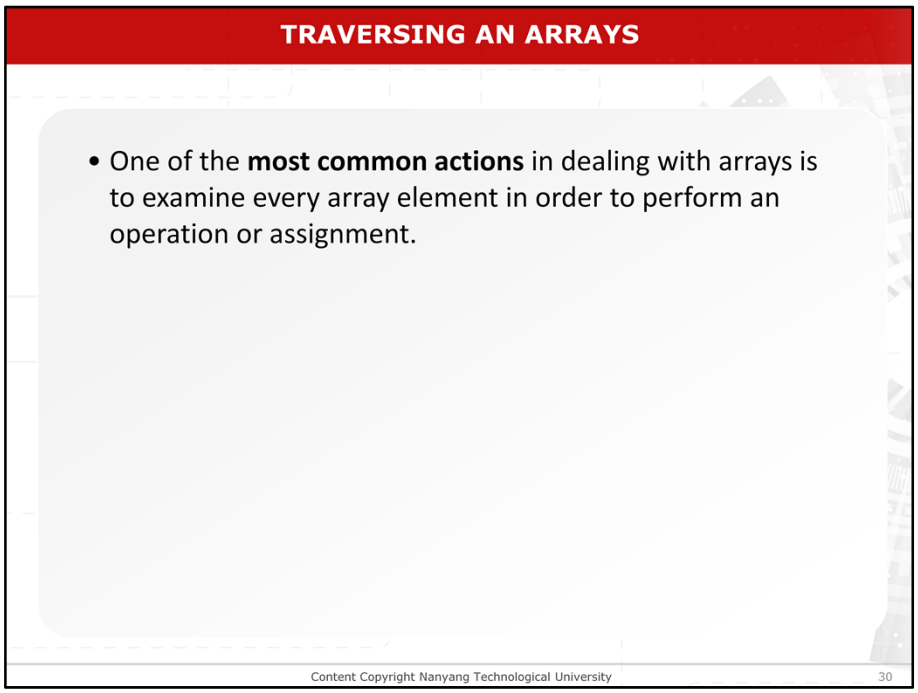

```
days[1] = 29;                          // valid
days[2] = days[2] + 4;                 // valid
days[3] = days[2] + days[3];           // valid
days[4] = {2,3,4,5,6};
```


Is this statement valid?

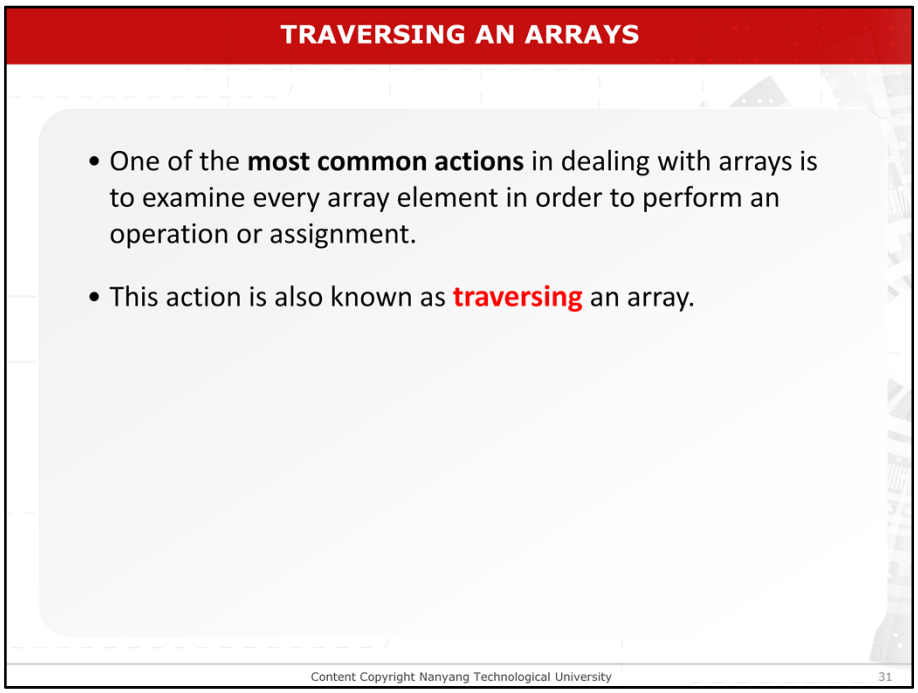
[pause for 10 sec]



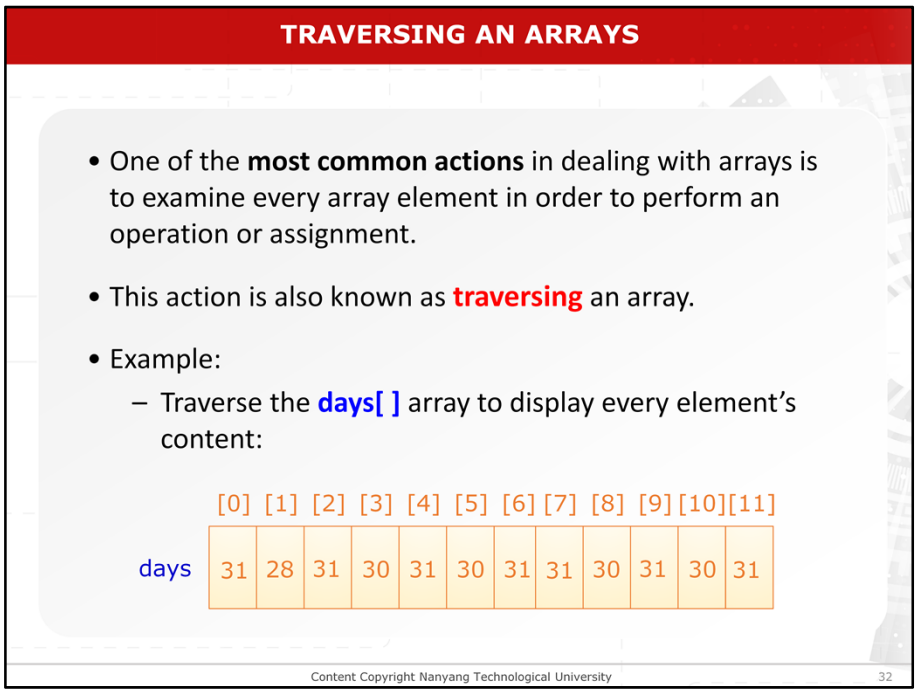
No. This statement is invalid because each element can only store an integer.



One of the **most common actions** in dealing with arrays is to examine every array element in order to perform an operation or assignment.



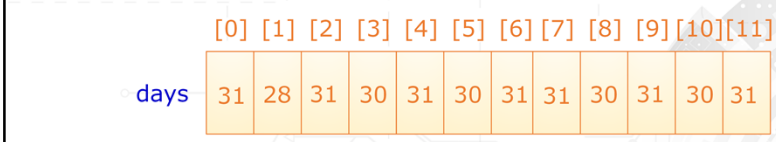
This action is also known as **traversing** an array.



For example, we want to traverse the **days** array to display every element's content.

Since array elements can be accessed individually, the most efficient way of manipulating array elements is to use a **for** or **while** loop. The loop control variable is used as the index for the array. Thus, each element of the array can be accessed as the value of the loop control variable changes when the loop is executed. Also note that array values are printed using the corresponding indexes.

EXAMPLE 1: PRINTING VALUES

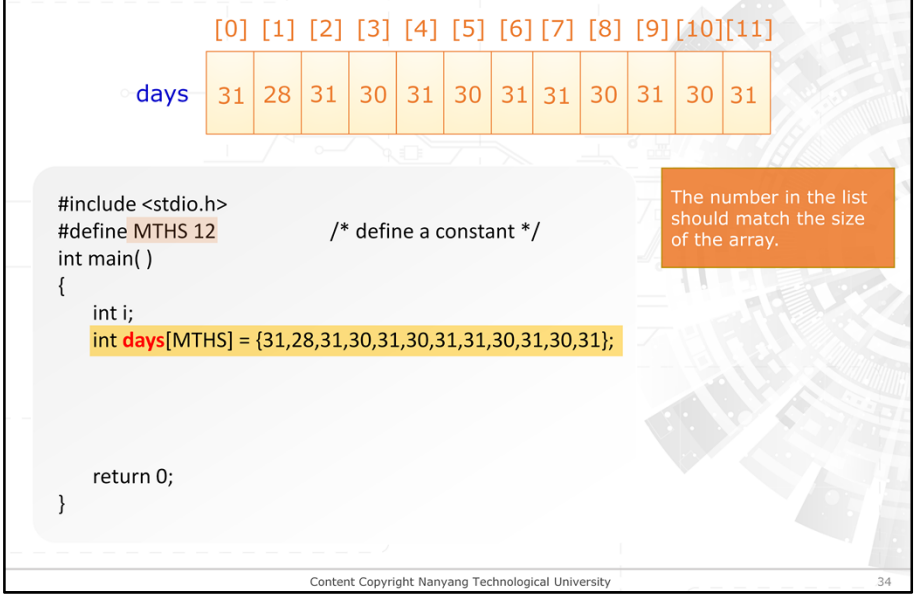



```
#include <stdio.h>
#define MTHS 12                 /* define a constant */
int main( )
{
    int i;
    int days[MTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};


    return 0;
}
```

The array **days** is first initialized using a list of integers.

In the program, the array **days** is first initialized using a list of integers.

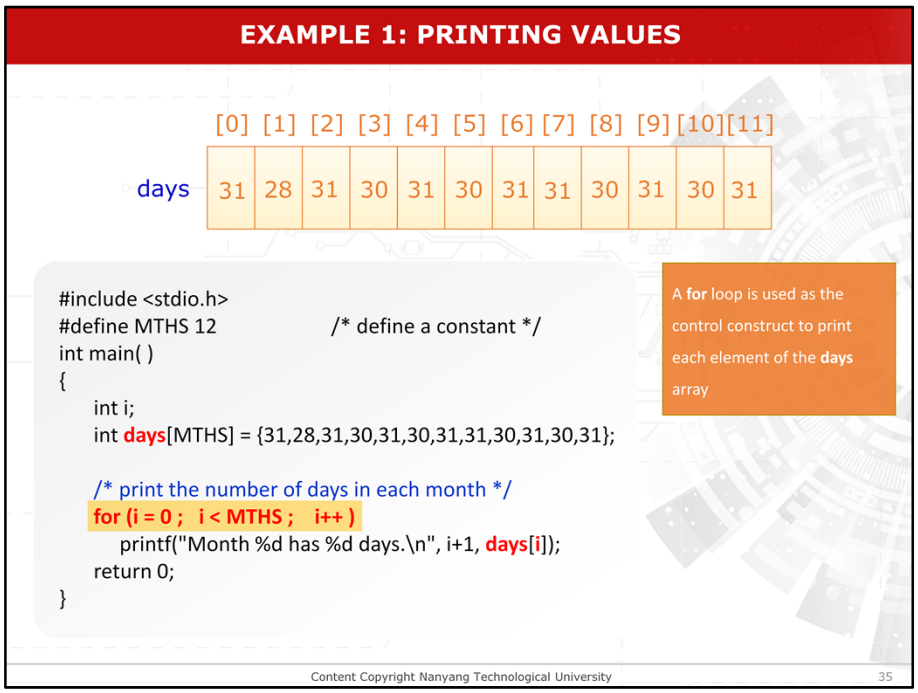
EXAMPLE 1: PRINTING VALUES



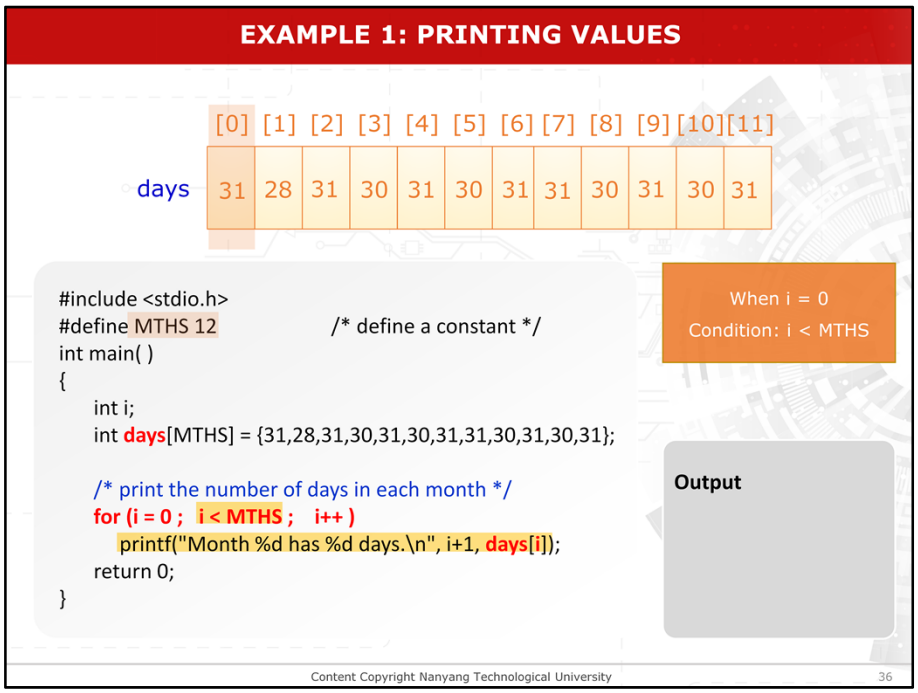

```
#include <stdio.h>
#define MTHS 12                 /* define a constant */
int main( )
{
    int i;
    int days[MTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};




    return 0;
}
```

The number in the list should match the size of the array.

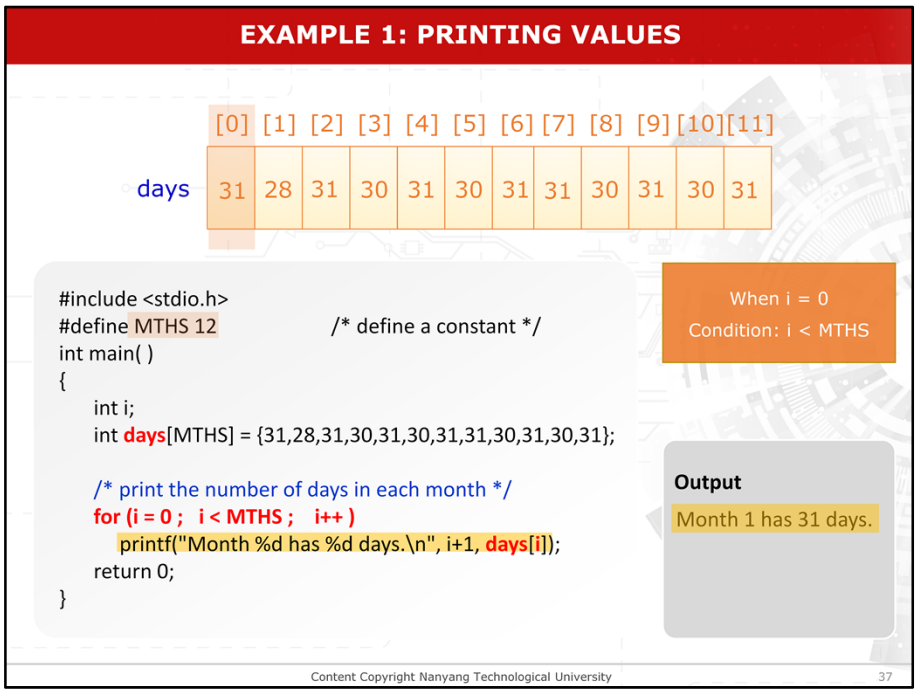
The number in the list should match the size of the array.
However, if the list is shorter than the size of the array, then the remaining elements are initialized to 0.



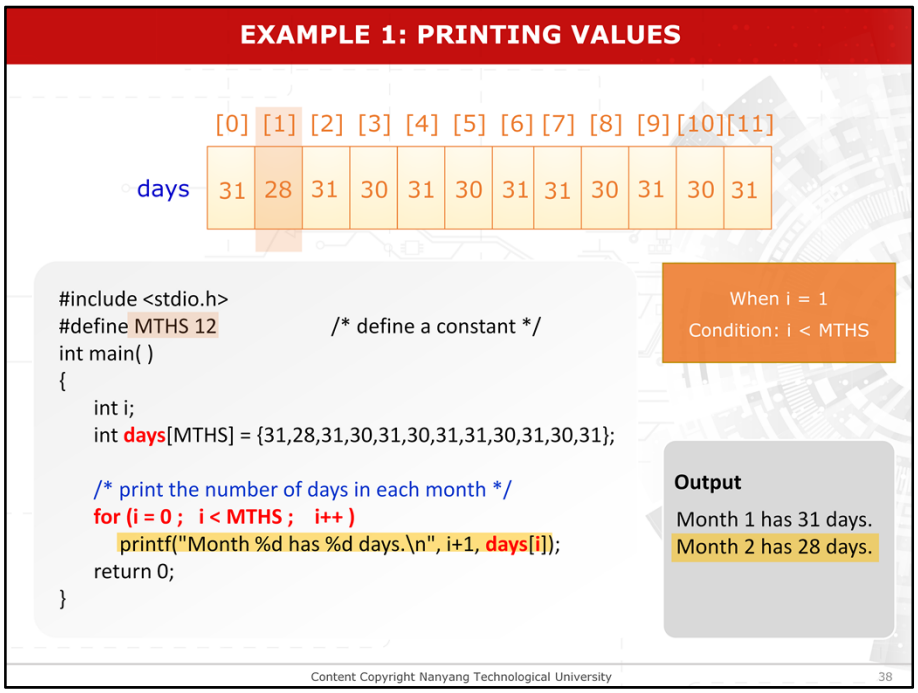
After that, a **for** loop is used as the control construct to print each element of the **days** array.



The for loop starts with I equals zero. It satisfies the condition of I lesser than Months which is defined as 12.

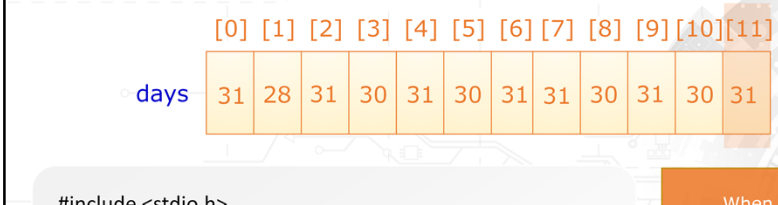


Thus the output is printed.



Then I is increased by 1 to I equals 1. The condition is again satisfied thus the output is printed.

EXAMPLE 1: PRINTING VALUES



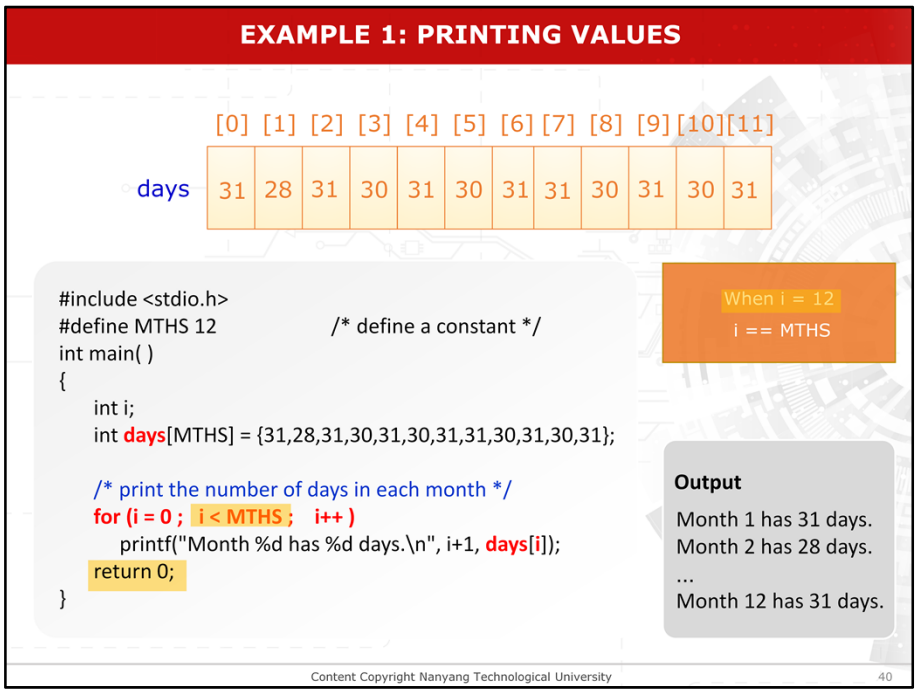

```
#include <stdio.h>
#define MTHS 12                 /* define a constant */
int main( )
{
    int i;
    int days[MTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};

    /* print the number of days in each month */
    for (i = 0 ;  i < MTHS ;   i++ )
        printf("Month %d has %d days.\n", i+1, days[i]);
    return 0;
}
```

When i = 11
Condition: i < MTHS

Output

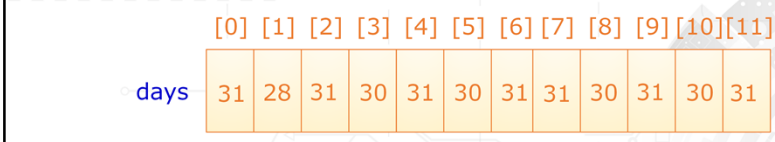
```
Month 1 has 31 days.
Month 2 has 28 days.
...
Month 12 has 31 days.
```


This keep repeating till I equals 11.



After I equals 11 increased by 1 to I equals 12, it no longer satisfied the for loop condition thus exited the loop.

EXAMPLE 1: PRINTING VALUES

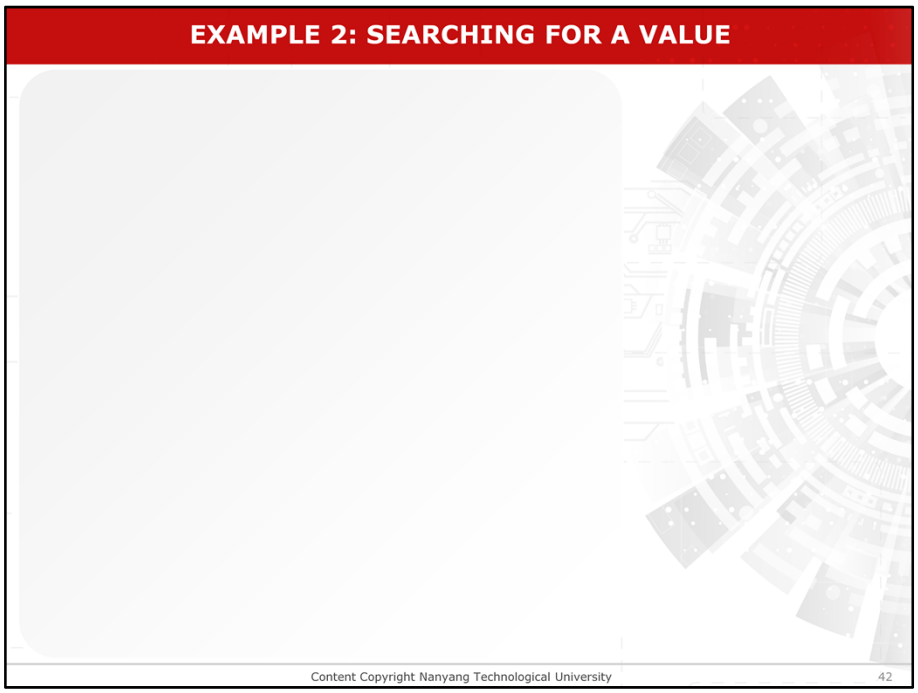



```
#include <stdio.h>
#define MTHS 12                 /* define a constant */
int main( )
{
    int i;
    int days[MTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};

    /* print the number of days in each month */
    for (i = 0 ;  i < MTHS ;  i++ )
        printf("Month %d has %d days.\n", i+1, days[i]);
    return 0;
}
```

Output

```
Month 1 has 31 days.
Month 2 has 28 days.
...
Month 12 has 31 days.
```

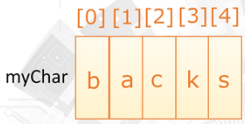

So in this example 1, we learnt how to traverse an array to print the values using a for loop.
In the next example, we will see how to traverse an array to search for a value.



Example 2: Searching for a value.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
    char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};




    return 0;
}
```

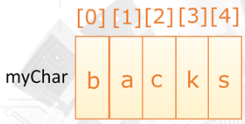


The element that needs to be found is called a ***search key***.

When working with arrays, it may be necessary to search for the presence of a particular element.
The element that needs to be found is called a *search key*.
In the program, the array **my Char** is first initialized using a list of characters.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
     char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
     int i;
     char searchChar;
     // Reading in user's input to search
     printf("Enter a char to search: ");
     scanf("%c", &searchChar);









     return 0;
}
```



Output

Enter a char to search:

The user can then enter the target character to search.

The user can then enter the target character to search.
The character entered will be stored in the variable search Char.
The program will then traverse the array to find the index position of the target character.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
     char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
     int i;
     char searchChar;
     // Reading in user's input to search
     printf("Enter a char to search: ");
     scanf("%c", &searchChar);

     // Traverse myChar array and output character if found
     for  (i = 0;    i < SIZE;     i++) {



     }
     return 0;
}
```

Output

Enter a char to search:

The program searches for the search key from the array **my Char** and returns the corresponding index position if found.
In the program, the target character is firstly read from the user.
Then, the character values stored in the array are checked one by one using a **for** loop.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
    char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
    int i;
    char searchChar;
    // Reading in user's input to search
    printf("Enter a char to search: ");
    scanf("%c", &searchChar);

    // Traverse myChar array and output character if found
    for  (i = 0;    i < SIZE;    i++) {
        if (myChar[i] == searchChar){
            printf ("Found %c at index %d", myChar[i], i);
            break;     //break out of the loop
        }
    }
    return 0;
}
```

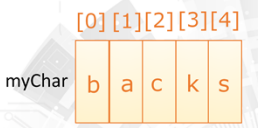
Output

Enter a char to search:

If the character value of the checked item is the same as the target character, the corresponding index position is then printed on the screen.
And the **break** statement is executed to exit the loop.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
     char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
     int i;
     char searchChar;
     // Reading in user's input to search
     printf("Enter a char to search: ");
     scanf("%c", &searchChar);

     // Traverse myChar array and output character if found
     for  (i = 0;    i < SIZE;     i++) {
          if (myChar[i] == searchChar){
               printf ("Found %c at index %d", myChar[i], i);
               break;     //break out of the loop
          }
     }
     return 0;
}
```



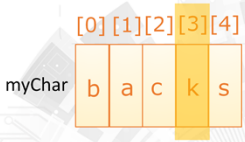
Output

Enter a char to search: k

For example, if character K is entered as the search key, the for loop is then used to compare with the characters stored in the array.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
    char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
    int i;
    char searchChar;
    // Reading in user's input to search
    printf("Enter a char to search: ");
    scanf("%c", &searchChar);

    // Traverse myChar array and output character if found
    for  (i = 0;    i < SIZE;    i++) {
        if (myChar[i] == searchChar){
            printf ("Found %c at index %d", myChar[i], i);
            break;     //break out of the loop
        }
    }
    return 0;
}
```



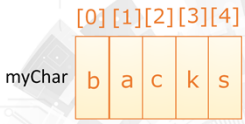
Output

Enter a char to search: k
Found k at index 3

It was found that the character K is at index position 3 thus the output is printed as shown. After which, the loop is exited.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
     char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
     int i;
     char searchChar;
     // Reading in user's input to search
     printf("Enter a char to search: ");
     scanf("%c", &searchChar);

     // Traverse myChar array and output character if found
     for  (i = 0;    i < SIZE;     i++) {
          if (myChar[i] == searchChar){
               printf ("Found %c at index %d", myChar[i], i);
               break;     //break out of the loop
          }
     }
     return 0;
}
```



Output

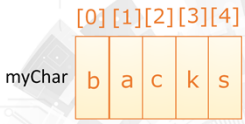
Enter a char to search: k
Found k at index 3

This linear search algorithm compares each element of the array with the search key until a match is found or the end of the array is reached.

This linear search algorithm compares each element of the array with the search key until a match is found or the end of the array is reached.
The program uses linear search by comparing each element of the array with the target character.
On average, the linear search algorithm requires to compare the search key with half of the elements stored in an array.

EXAMPLE 2: SEARCHING FOR A VALUE

```
#include <stdio.h>
#define SIZE 5                    /* define a constant */
int main ( )
{
     char myChar[SIZE] = {'b', 'a', 'c', 'k', 's'};
     int i;
     char searchChar;
     // Reading in user's input to search
     printf("Enter a char to search: ");
     scanf("%c", &searchChar);

     // Traverse myChar array and output character if found
     for  (i = 0;    i < SIZE;     i++) {
          if (myChar[i] == searchChar){
               printf ("Found %c at index %d", myChar[i], i);
               break;     //break out of the loop
          }
     }
     return 0;
}
```



Output

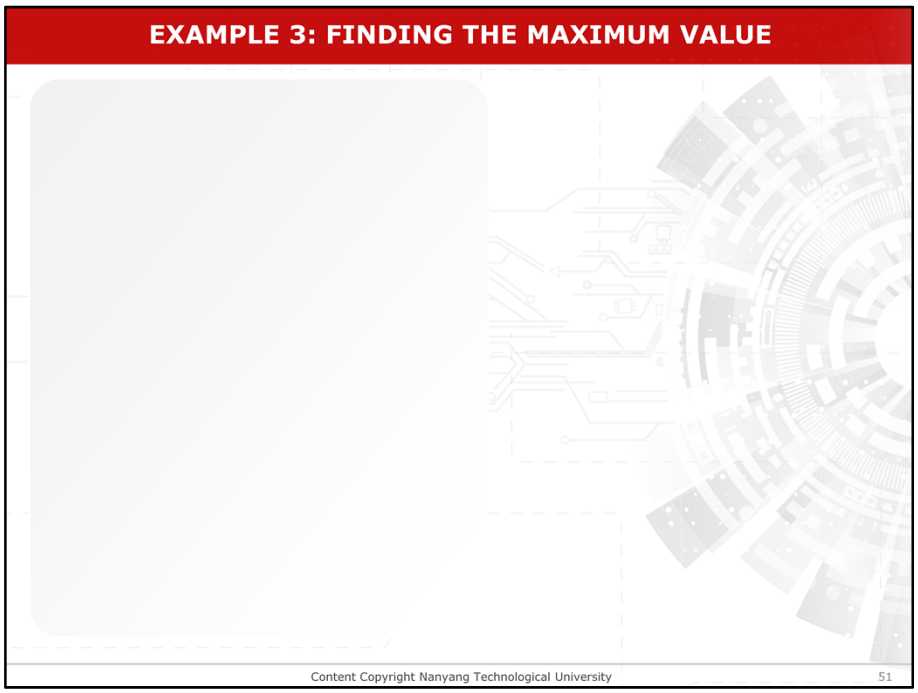
Enter a char to search: k
Found k at index 3

Linear search is sufficient for small arrays.

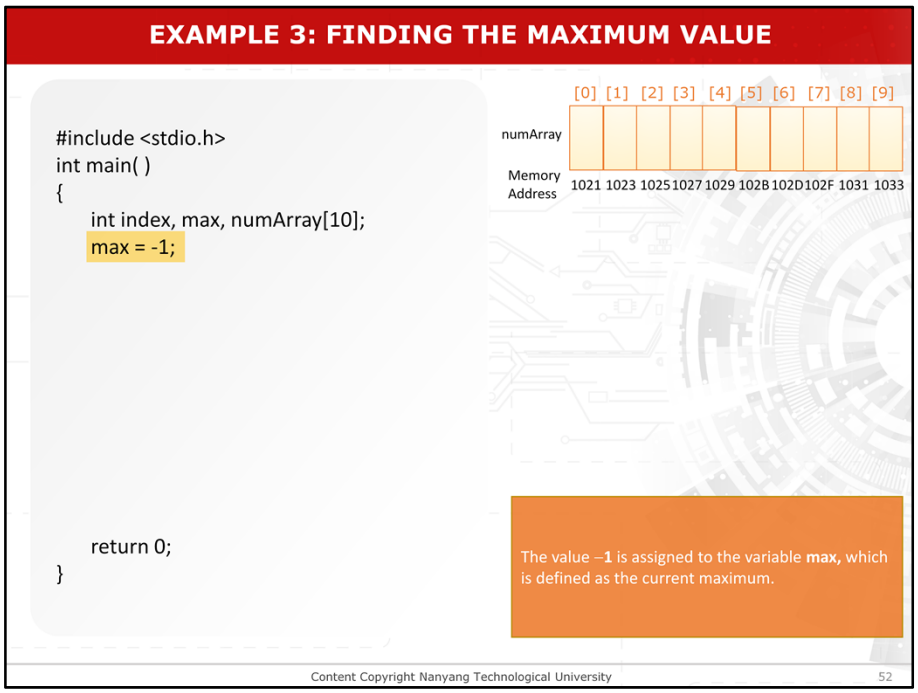
However, it is inefficient for large and sorted arrays.

Linear search is sufficient for small arrays.
However, it is inefficient for large and sorted arrays.
Therefore, a more efficient technique such as binary search should be used for large arrays.

In the next example, we will see how to traverse an array to find the maximum value.



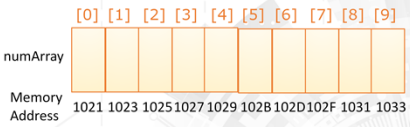
Example 3: Finding the maximum value.
The program finds the maximum non-negative value in an array.



In the program, the value for each item in an array is firstly read from the user and stored in the array.
The value –**1** is assigned to the variable **max,** which is defined as the current maximum.

EXAMPLE 3: FINDING THE MAXIMUM VALUE

```
#include <stdio.h>
int main( )
{
    int index, max, numArray[10];
    max = -1;
    printf("Enter 10 positive numbers: \n");




    return 0;
}
```



Output

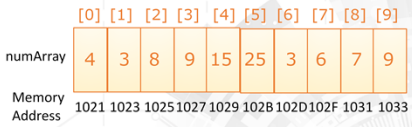
Enter 10 positive numbers:

User is asked to input 10 numbers.

User is asked to input 10 **positive** numbers.

EXAMPLE 3: FINDING THE MAXIMUM VALUE

```
#include <stdio.h>
int main( )
{
    int index, max, numArray[10];
    max = -1;
    printf("Enter 10 positive numbers: \n");
    for (index = 0; index < 10; index++)
        scanf("%d", &numArray[index]);

    return 0;
}
```



Output

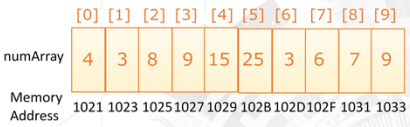
Enter 10 positive numbers:
4 3 8 9 15 25 3 6 7 9

The input numbers are stored in the array declared.

The input numbers are stored in the array declared.

EXAMPLE 3: FINDING THE MAXIMUM VALUE

```
#include <stdio.h>
int main( )
{
    int index, max, numArray[10];
    max = -1;
    printf("Enter 10 positive numbers: \n");
    for (index = 0; index < 10; index++)
        scanf("%d", &numArray[index]);

    // Find maximum from array data
    for (index = 0;   index < 10;    index++) {


    }

    return 0;
}
```

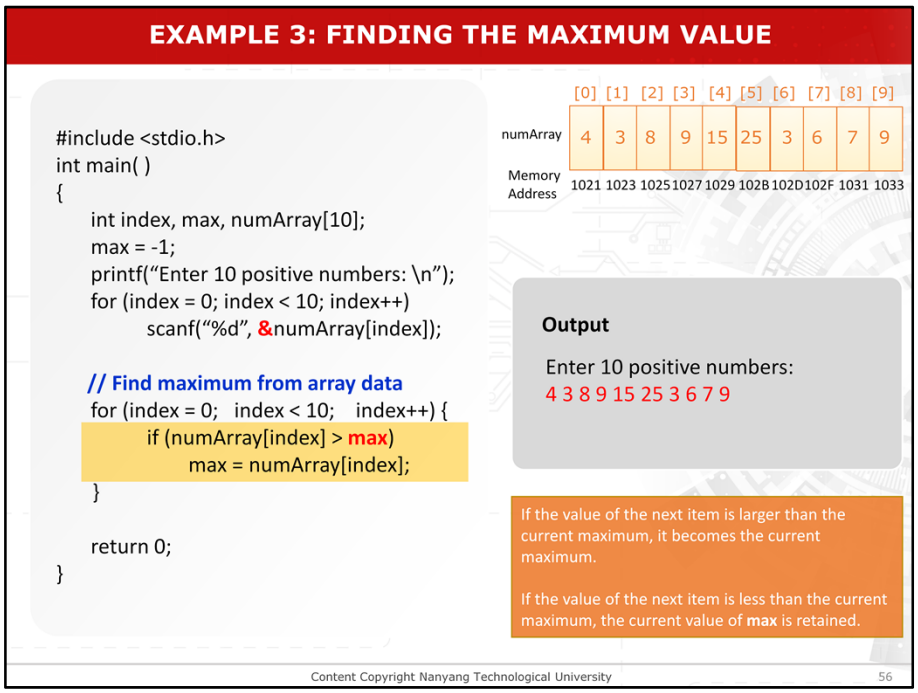


Output

Enter 10 positive numbers:
4 3 8 9 15 25 3 6 7 9

Then, the items in the array are checked one by one using a **for** loop.

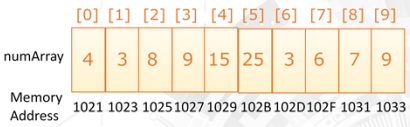
Then, the items in the array are checked one by one using a **for** loop.



If the value of the next item is larger than the current maximum, it becomes the current maximum.
If the value of the next item is less than the current maximum, the current value of **max** is retained.

EXAMPLE 3: FINDING THE MAXIMUM VALUE

```
#include <stdio.h>
int main( )
{
    int index, max, numArray[10];
    max = -1;
    printf("Enter 10 positive numbers: \n");
    for (index = 0; index < 10; index++)
        scanf("%d", &numArray[index]);

    // Find maximum from array data
    for (index = 0;  index < 10;  index++) {
        if (numArray[index] > max)
            max = numArray[index];
    }
    printf("The max value is %d.\n", max);
    return 0;
}
```

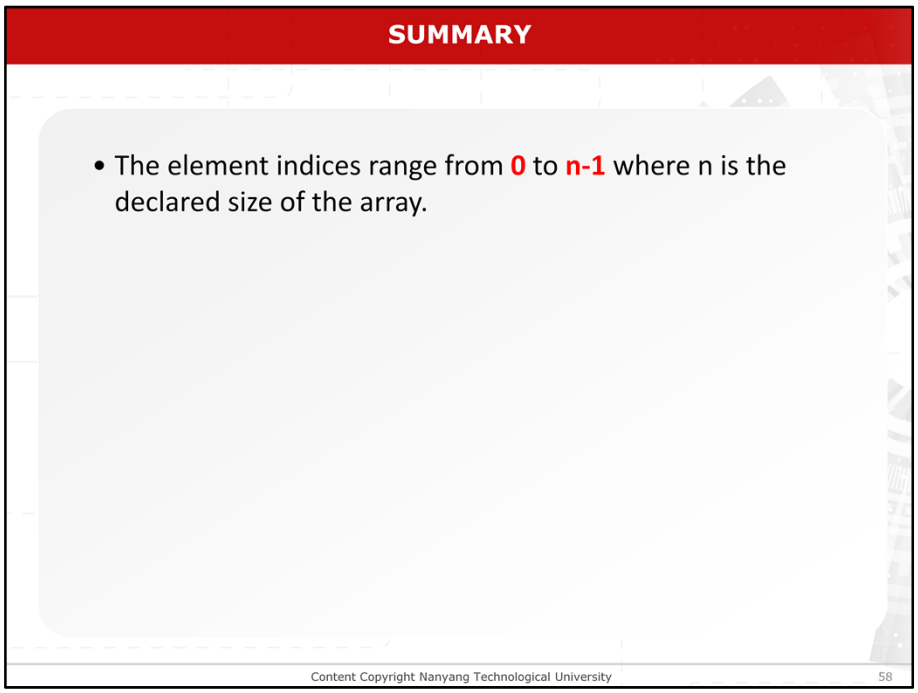


Output

Enter 10 positive numbers:
4 3 8 9 15 25 3 6 7 9
The max value is 25.

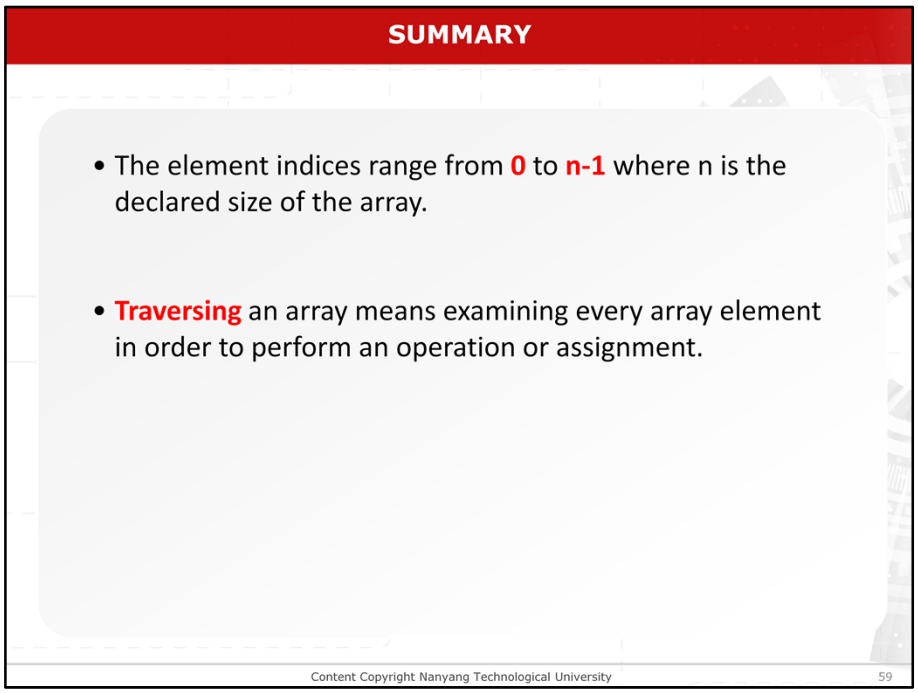
The maximum value in the array is then printed on the screen.

The maximum value in the array is then printed on the screen.

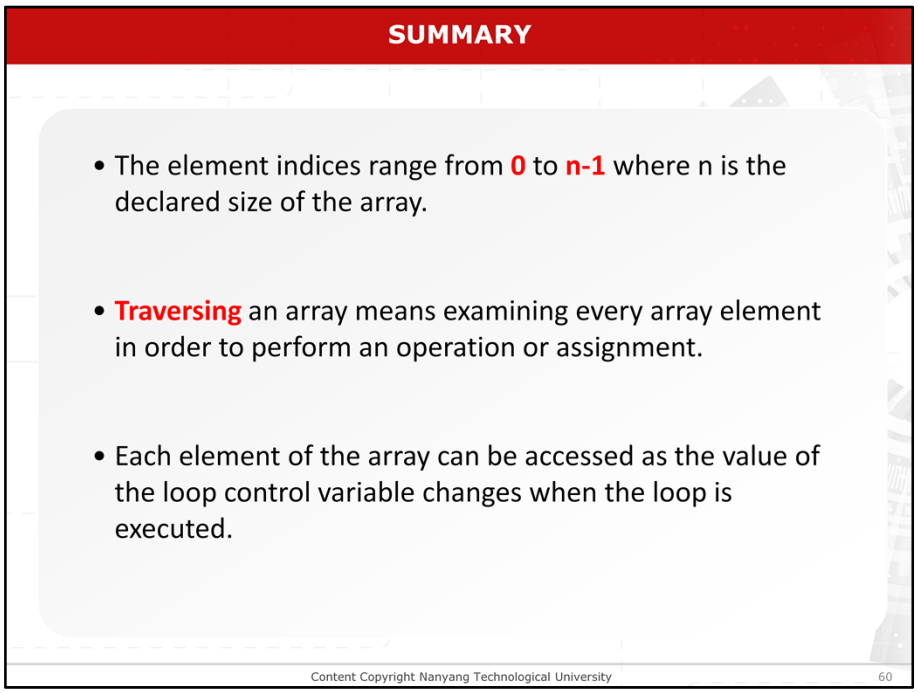


Let's do a quick summary on the operation on array.

First, the element indices range from **0** to **n minus 1** where n is the declared size of the array.



Next, **traversing** an array means examining every array element in order to perform an operation or assignment.



Lastly, each element of the array can be accessed as the value of the loop control variable changes when the loop is executed.