5.1
One-dimensional Arrays

Why Learning Arrays?

- Most programming languages provide array data structure as built-in data structure.
- An array is a list of values with the **same** data type that can be used to organize and store related data items. If not using array, you will need to define many variables instead of just **one** array variable.
- Python provides the **list** structure, which has two major differences from the array data structure in C:
  - Arrays have only limited operations while lists have many operations.
  - Size of arrays cannot be changed while lists can grow and shrink.
- In arrays, we can categorize them as one-dimensional arrays and two-dimensional (or multi-dimensional) arrays. In this lecture, we focus on discussing one-dimensional arrays.

One-dimensional Arrays

- **Array Declaration, Initialization and Operations**
- Pointers and Arrays
- Arrays as Function Arguments

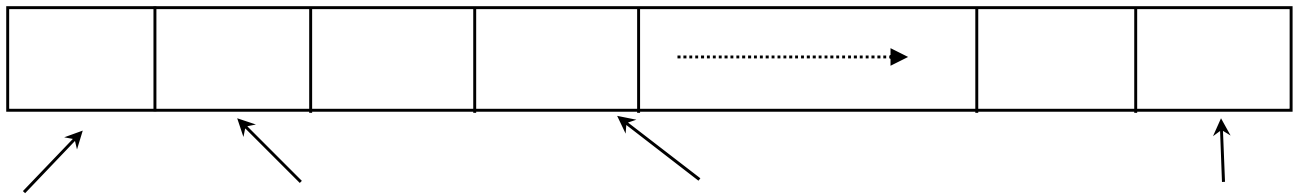
Types of Variables

- Data (or values) stored in variables are mainly in two forms:
  - **Primitive Variables**: Variables that are used to store **values**. They are mainly variables of primitive data types, such as int, float and char. Later on, you will learn **Structure**, which is used to store a record of data (values).
  - **Reference (or Pointer) Variables**: Variables that are used to store **addresses**, such as **pointer variables, array variables** and **string variables**.

What is an Array?

- An **array** is a **list of values** with the **same data type**. Each value is stored at a specific, numbered position in the array.
- An array uses an **integer** called **index** to reference an element in the array.
- The **size** of an array is **fixed** once it is created. Could the size be created dynamically? Yes by using **malloc()**, you will learn that later in data structures.
- Index always starts with **0** **(zero)**.

Array of size 12



1st element index=**0**

2nd element index=1

Nth element index=N-1

12th element index=**11**

Array Declaration

- Declaration of arrays **without initialization**:

```
char name[12];          /* array of 12 characters */
float sales[365];       /* array of 365 floats */
int states[50];         /* array of 50 integers */
int *pointers[5];       /* array of 5 pointers to integers */
```

- When an array is declared, some **consecutive memory** locations are allocated by the compiler for the whole array (**2 or 4** bytes will be allocated for an integer depending on machine):

  total_memory = **sizeof**(type_specifier)*array_size;

  e.g. char name[12]; - total_memory = 1*12 = 12 bytes

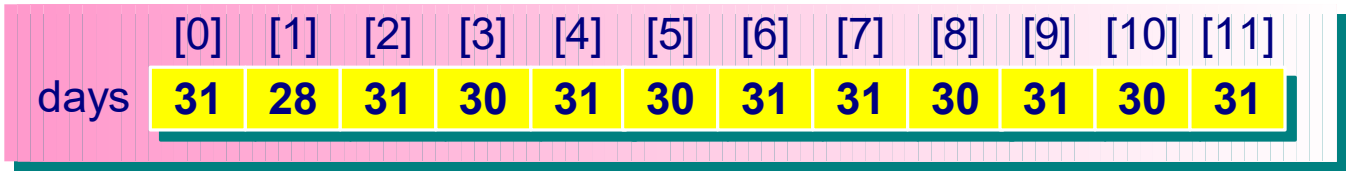
- The size of array must be integer constant or constant expression in declaration:

```
e.g.    char name[i];     // i is a variable  ==> illegal
        int states[i*6];  // i is a variable  ==> illegal
```

Initialization of Arrays

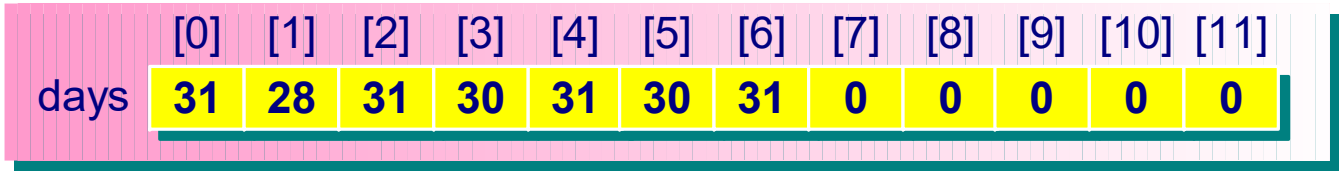
- **Initialize array variables** at declaration:

```
int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};
```



- **Partial array initialization**: E.g. (initialize first 7 elements)

```
int days[12]={31,28,31,30,31,30,31};
/* remaining elements are initialized to zero */
```



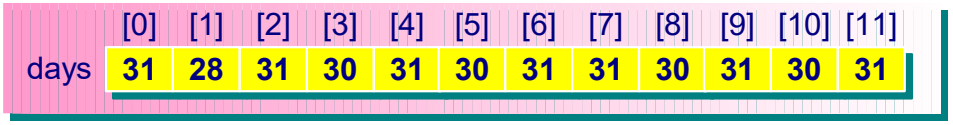
Operations on Arrays

- **Accessing** array elements:

```
sales[0] = 143.50;          // using array index
if (sales[23] == 50.0) ...
```

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

```
char name[12];
name[12] = 'c';          // index out of range – common error
```

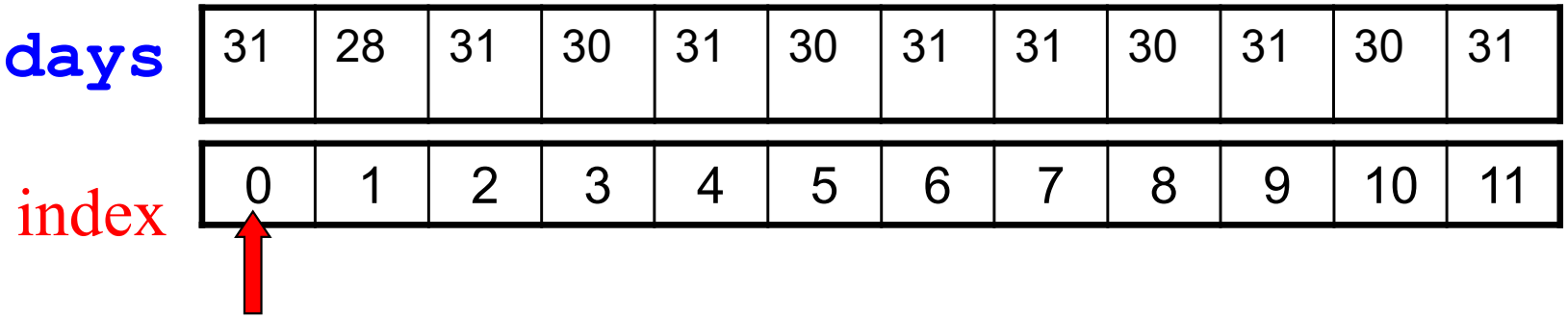


- Working on array values:

```
(1) days[1] = 29;                    - OK ??
(2) days[2] = days[2] + 4;           - OK ??
(3) days[3] = days[2] + days[3]; - OK ??
(4) days[1] = {2,3,4,5,6};           - OK? NOT OK!!
```

Traversing an Array – Using Array Index

- 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.
- Example:
  - Traverse the **days[ ]** array using a **for** or **while** loop to access each array element individually with array index, and then process each array element's content accordingly.

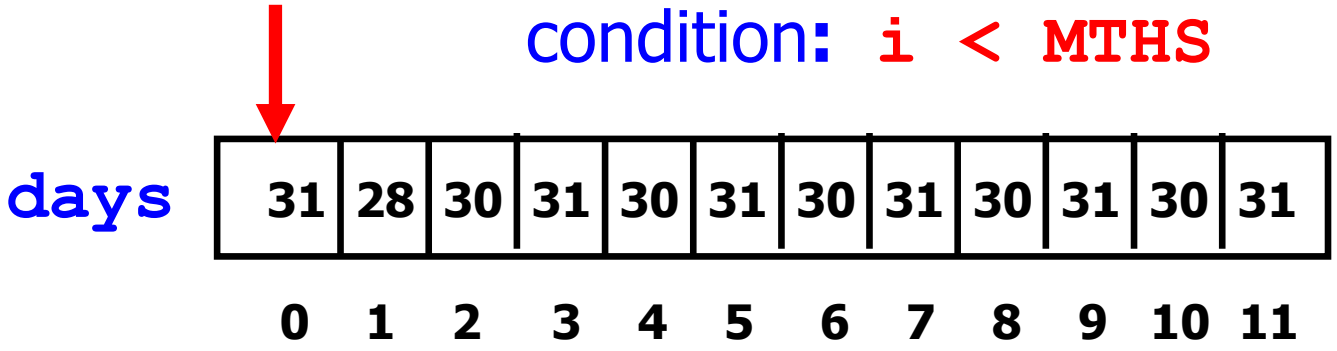


Example 1: Printing Values

```
#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.



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: ***a***

Found a at index 1

Example 3: Finding the Maximum Value

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

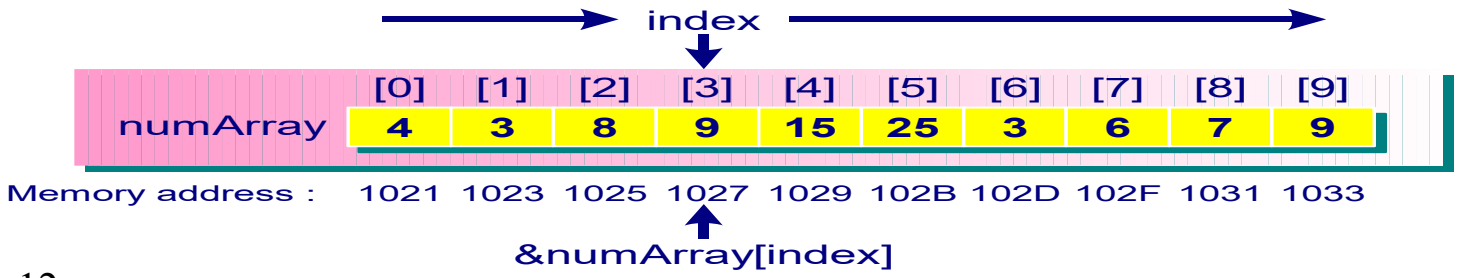
This example shows how to find the largest value in an array of numbers.

Output

Enter 10 numbers:

4 3 8 9 15 25 3 6 7 9

The max value is 25.



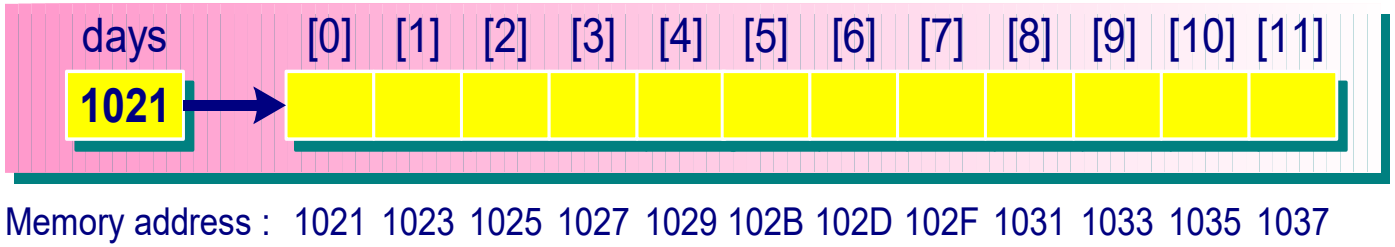
One-dimensional Arrays

- Array Declaration, Initialization and Operations
- **Pointers and Arrays**
- Arrays as Function Arguments

Pointer Constants

- The **array name** is actually a ***pointer constant***.

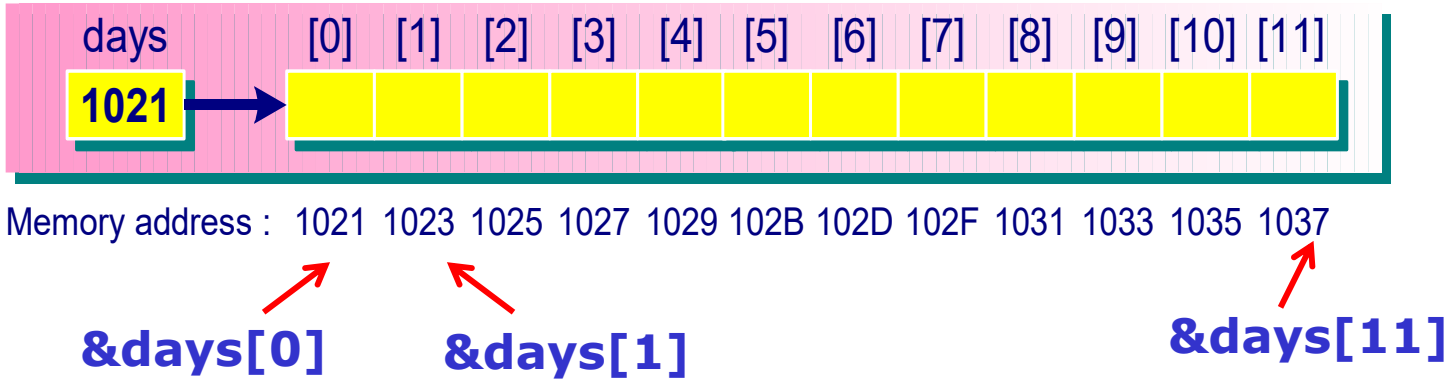
  e.g. int **days**[12]; // **days** – **pointer constant**



- The array **days** begins at memory location 1021. Here, we use 2 bytes to represent an integer value (for older machines) for illustration purpose. Note that most current systems represent an integer using 4 bytes.

Pointer Constants (Cont'd.)

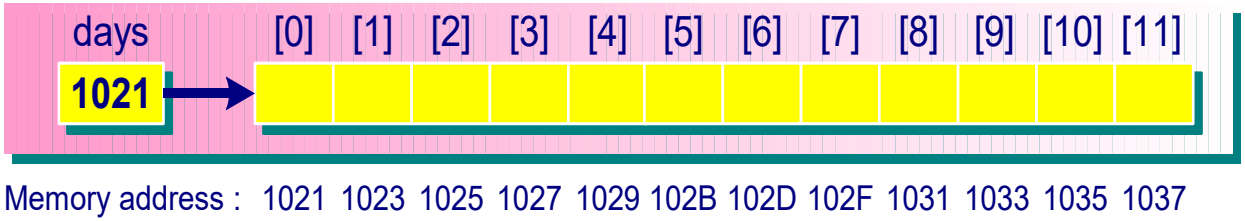
- Address of an array element:



&days[0] - is the **address** of the **1st** element [i.e. 1021]
&days[1] - is the **address** of the **2nd** element [i.e. 1023]
&days[i] - is the **address** of the **(i+1)th** element

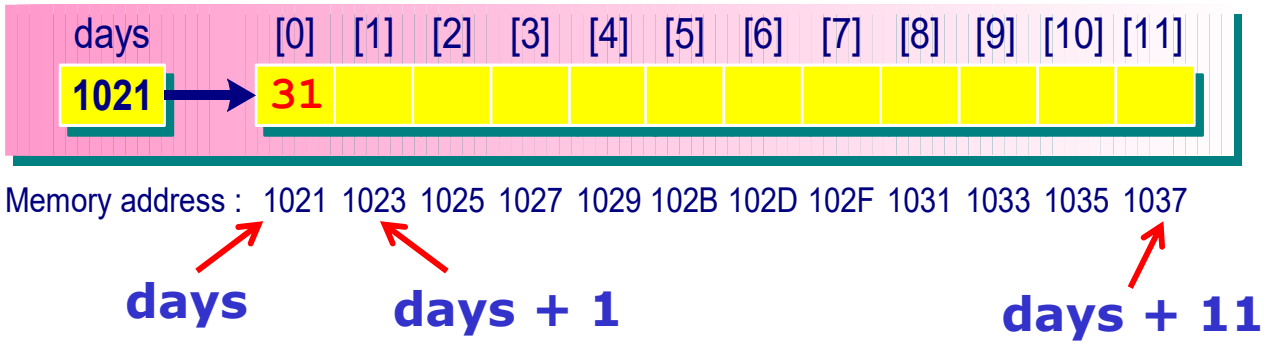
Pointer Constants (Cont'd.)

- **days** - is the **address (or pointer)** of the **1st element of the array**



- Note:
  - Array variable: **days** – contains a pointer constant (i.e. 1021) (the value cannot be changed)
  - Array with index: **days[0]**, days[1], etc. – contains the array value at that index location
  - Array element address: **&days[0] (i.e. 1021)**, &days[1], etc. - days[0] has the address of 1021, days[1] has the address of 1023, etc.
- Can we use the pointer **days** for accessing each array element?

Pointer Constants (Cont'd.)



- To do that, we need to know two important concepts:

(1) **array_name** (i.e. pointer constant)

days == &days[0] (i.e. 1021)

days + i == &days[i]

(2) ***array_name (dereferencing)**

***days** == days[0] (i.e. 31)

***(days + i)** == days[i]

Note: You may also use ***days** to refer to the content stored at **days[0]**, etc.

- But, you **cannot** change the array **base pointer**:

```
days += 5;     // i.e. days = days+5;     not valid
days++;        // i.e. days = days+1;     not valid
```

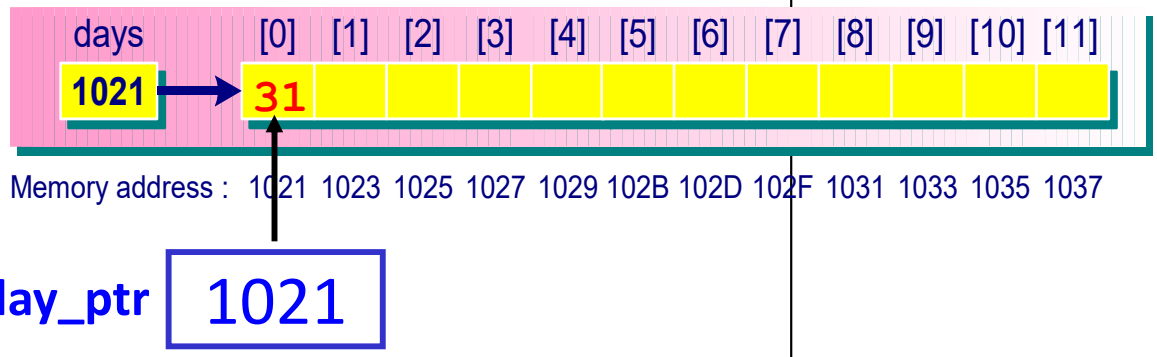
Pointer Variables

- A ***pointer variable*** can take on **different addresses**.

```
/* pointer arithmetic */
#define MTHS 12
int main()
{
    int days[MTHS]= {31,28,31,30,31,30,31,31,30,31,30,31};
    int *day_ptr;

1 . day_ptr = days;

    printf("First element = %d\n", *day_ptr);
}
```

Pointer constant

Pointer variable



Output

First element = 31

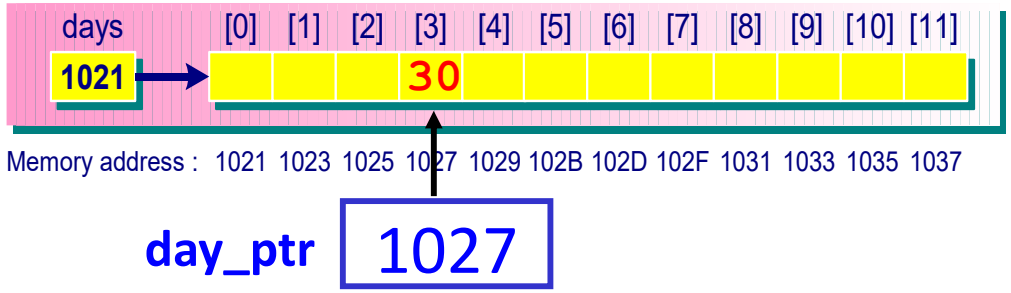
Pointer Variables (Cont'd.)

- A ***pointer variable*** can take on **different addresses**.

```
/* pointer arithmetic */
#define MTHS 12
int main()
{
    int days[MTHS]= {31,28,31,30,31,30,31,31,30,31,30,31};
    int *day_ptr;
    day_ptr = days;
    printf("First element = %d\n", *day_ptr);
2.  day_ptr = &days[3]; /* points to the fourth element */
    printf("Fourth element = %d\n", *day_ptr);
}
```

Pointer constant

Pointer variable

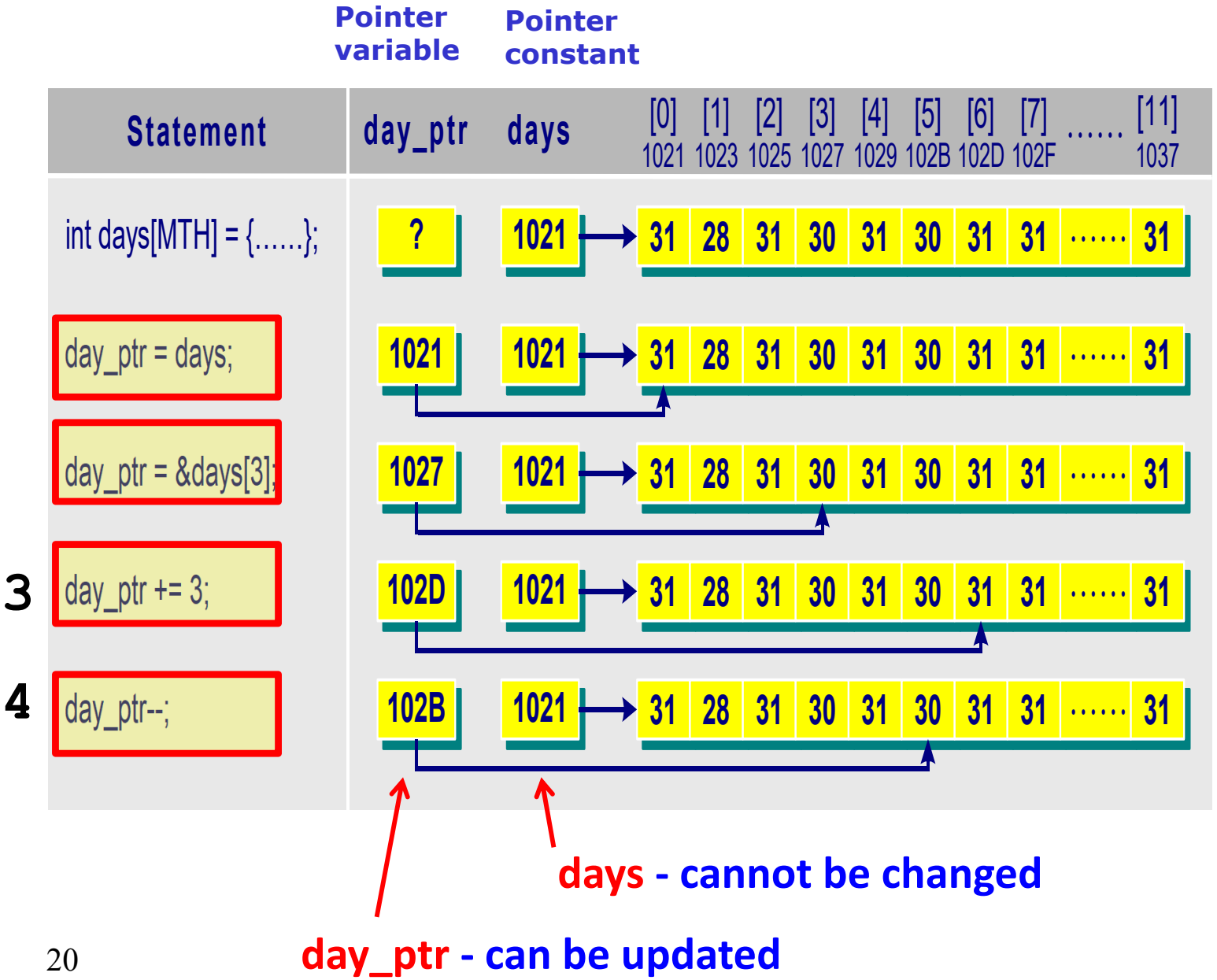


Output

First element = 31

Fourth element = 30

Pointer Variables (Cont'd.)



Output

First element = 31
Fourth element = 30
Seventh element = 31
Sixth element = 30

Finding Maximum: Using Pointer Constants

```
#include <stdio.h>
int main( )
{
    int index, max, numArray[10];
    printf("Enter 10 numbers: \n");
    for (index = 0;  index < 10;  index++)
        scanf("%d", numArray + index);
    // Find maximum from array data
    max = *numArray;
    for (index = 1;  index < 10;  index++)
    {
        if (*(numArray + index) > max)
            max = *(numArray + index);
    }
    printf("The max value is %d.\n", max);
    return 0;
}
```

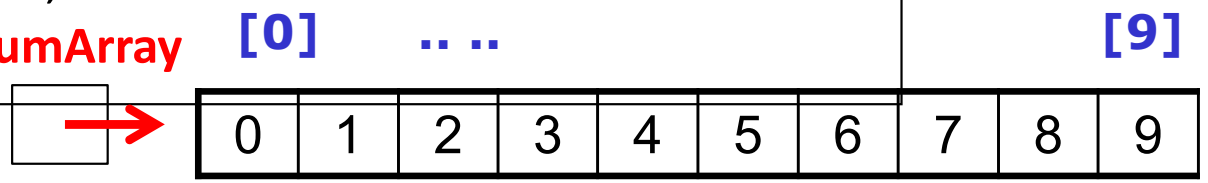
Pointer constant

Using index for reading input:

```
for (index = 0; index < 10; index++)
    scanf("%d", &numArray[index]);
```

Using index for processing:

```
max = numArray[0];
for (index = 1;  index < 10;
                index++)
{
    if (numArray[index] > max)
        max = numArray[index];
}
```



Finding Maximum: Using Pointer Variables

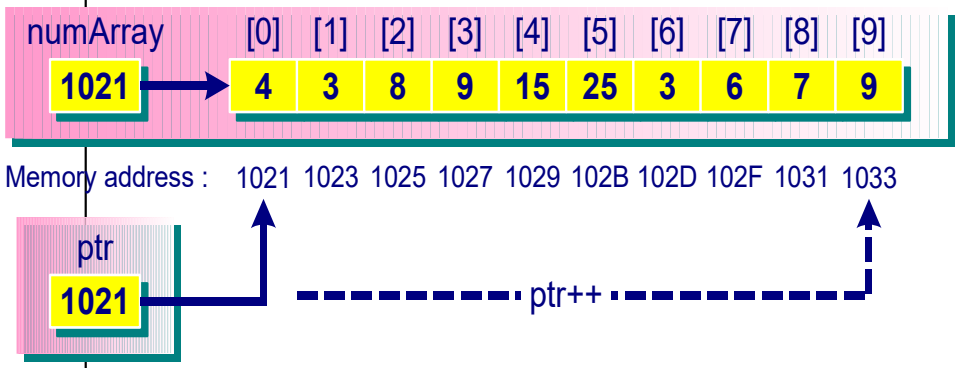
```
#include <stdio.h>
int main( ){
    int index, max, numArray[10];
    int *ptr;
    ptr = numArray;
    printf("Enter 10 numbers: \n");
    for (index = 0; index < 10; index++)
        scanf("%d", ptr++);
     // Find maximum from array data
    ptr = numArray;
    max = *ptr;
    for (index = 0; index < 10; index++) {
        if (*ptr > max)
            max = *ptr;
        ptr++;
    }
    printf("max is %d.\n", max);
    return 0;
}
```

Output

Enter 10 numbers:

4 3 8 9 15 25 3 6 7 9

max is 25.



Arrays and Pointers – Key Points

- Array is declared as **pointer constant**: In this case, we cannot change the base pointer address.
  - Example: int **numArray**[10];
  - Generally, we can use the **index notation** to access each element of the array, e.g. **numArray[0]** refers to the first element, etc.
  - We can also use the pointer constant to access each element of the array, e.g. ***(numArray+1)** refers to **numArray[1]**, etc. in order to access each element of the array.

- In addition, we can also declare **pointer variables** to access the array.
  - Declare a pointer variable and assign the array to the pointer variable. Example: int ***ptr**; **ptr = numArray;**
  - Then we can use **ptr** to access each element of the array.
  - For example, by dereferencing the pointer variable, ***ptr** refers to the first element of the array **numArray[0]**, etc. By updating the pointer variable (**ptr++**) to point to the next array element, we can then access each element of the array.

One-Dimensional Arrays

- Array Declaration, Initialization and Operations
- Pointers and Arrays
- **Arrays as Function Arguments**

Arrays as Function Arguments: Function Header

Function header

```
void fn1(int table[ ], int size)
{
   ....
}
```

or

```
void fn2(int table[TABLESIZE])
{
   ....
}
```

or

```
void fn3(int *table, int size)
{
   ....
}
```

The **prototype** of the function:

```
void fn1(int table[ ], int size);
```

or

```
void fn2(int table[TABLESIZE]);
```

or

```
void fn3(int *table, int size);
```

Note: **size** and **TABLESIZE** are the **data size** to be processed in the array

Arrays as Function Arguments: Calling the Function

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

```
fn1(table, n);        /* calling a function */
```

  where **fn1()** is a function and **table** is an one-dimensional 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.

Array as a Function Argument: Maximum

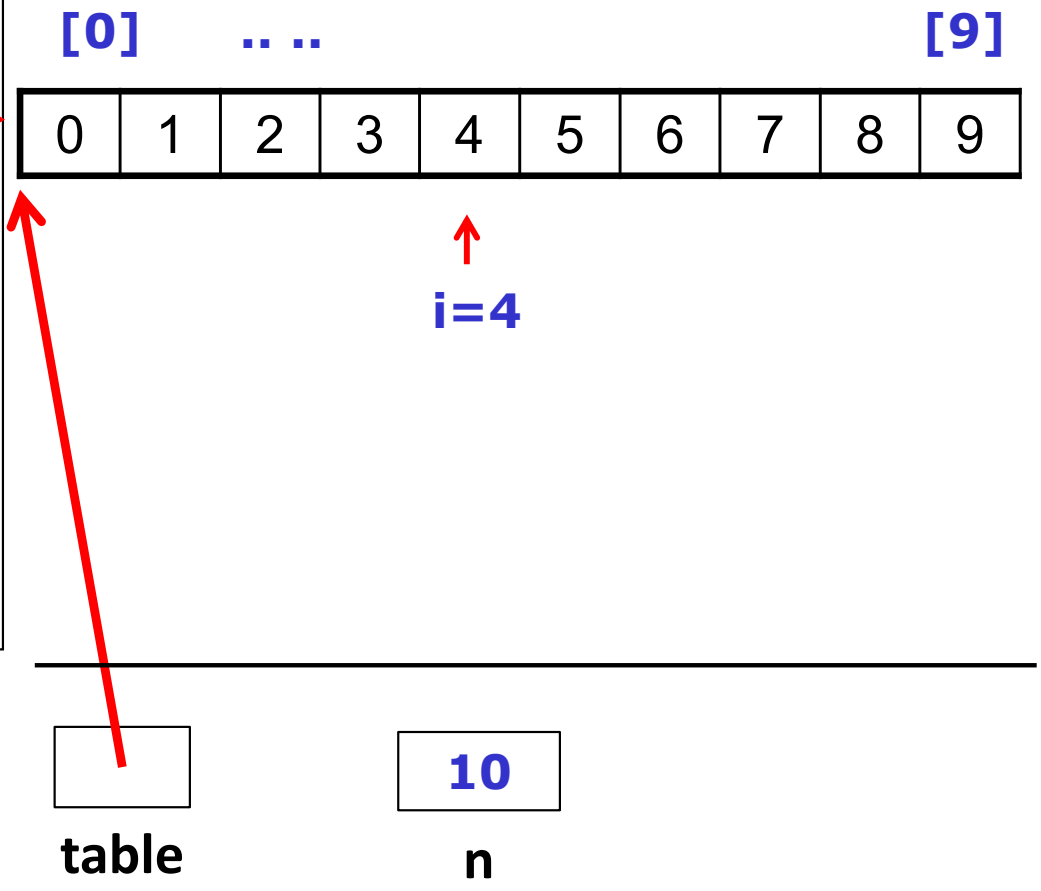
```
#include <stdio.h>
int maximum(int table[ ], int n);
int main( )
{
    int max, index, n;
    int numArray[10];   // Using index for input

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

    return 0 ;
}
```

Output

Enter the number of values: *10*
Enter 10 values: *0 1 2 3 4 5 6 7 8 9*
The maximum value is 9

Implementing Maximum: (1) Using Array Indexing

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

      ....
      max = maximum(numArray, n);
      printf("The maximum value is %d\n", max);
     return 0 ;
}
```

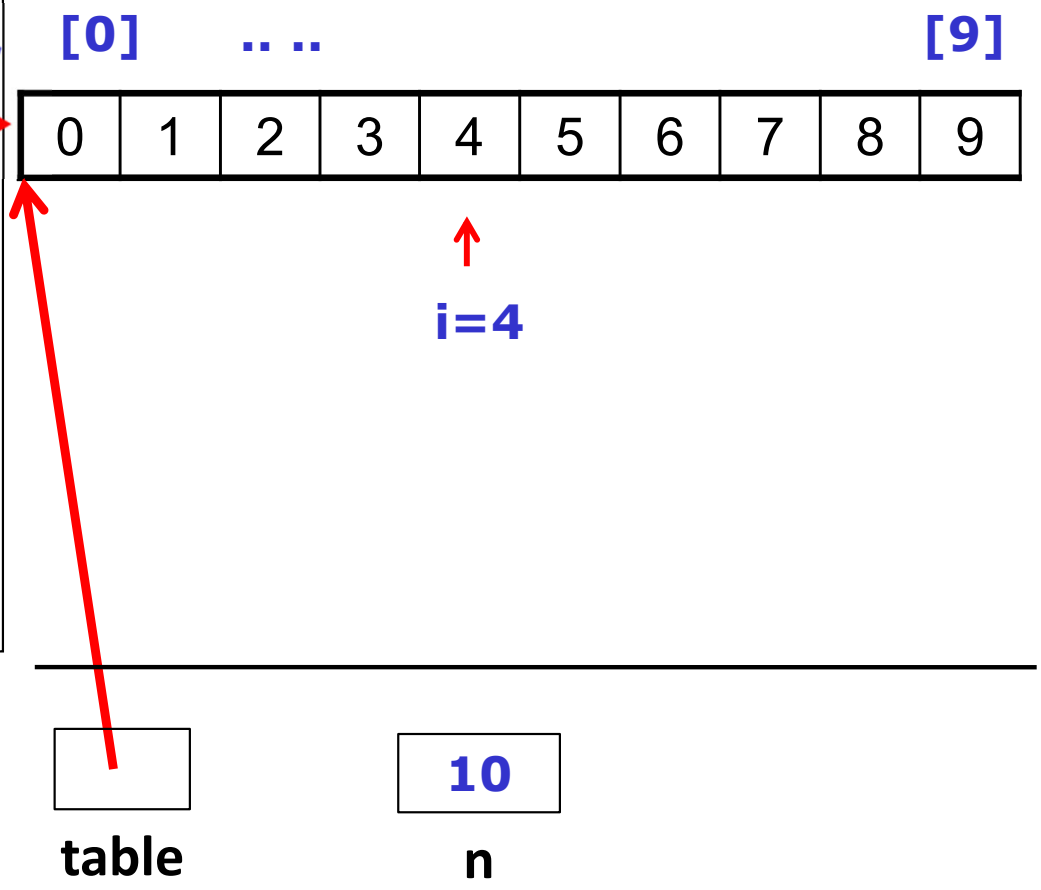



```
int maximum(int table[ ], int n)
{
     int i, max;

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

Using array indexing

Implementing Maximum: (2) Using Ar Base Address

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

      ....
      max = maximum(numArray, n);
      printf("The maximum value is %d\n", max);
     return 0 ;
}
```

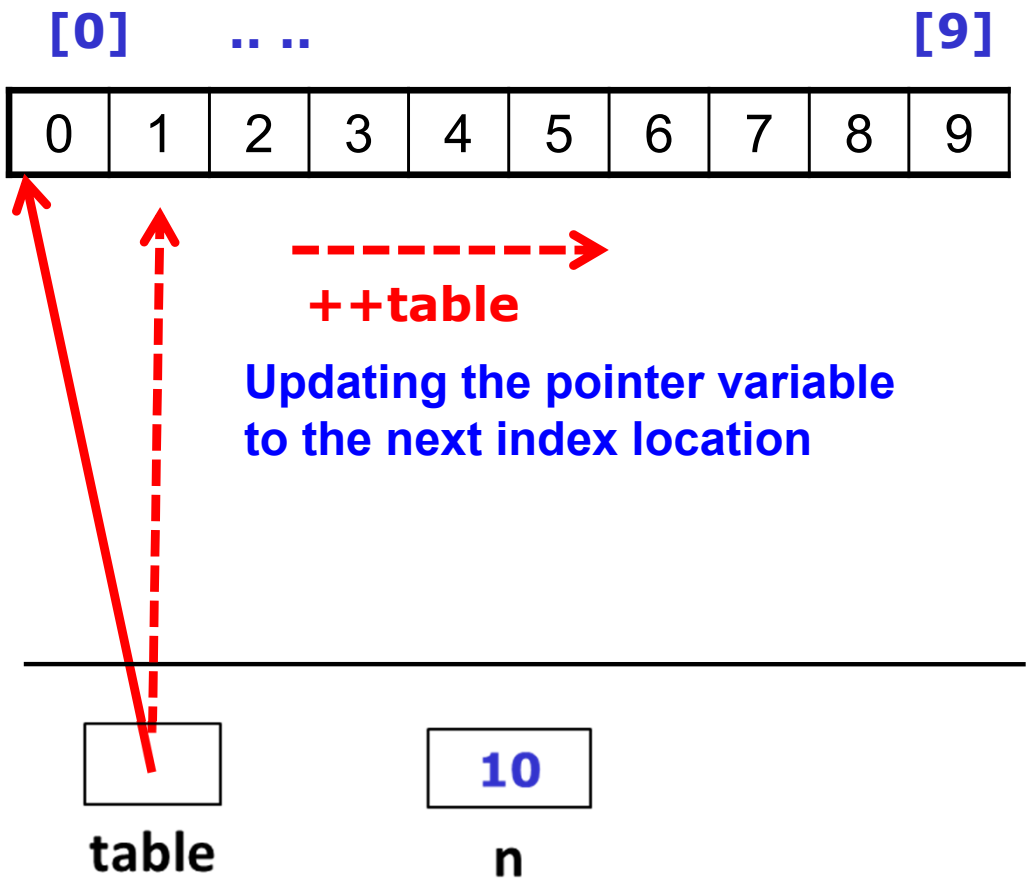



```
int maximum(int table[ ], int n)
{
     int i, max;

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

Using array base address

Implementing Maximum: (3) Using Pointer Variable

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

      ....
      max = maximum(numArray, n);
      printf("The maximum value is %d\n", max);
     return 0 ;
}
```




```
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;
}
```

Using pointer variable

Thank You!