Jiangxi University of Science and Technology

Chapter 11 Arrays, Addresses, and Pointers

- Lecture1102 Arrays and Pointers

Objectives

- 11.1 Array Names as Pointers
- 11.2 Manipulating Pointers
- 11.3 Passing and Using Array Addresses
- 11.4 Processing Strings Using Pointers
- 11.5 Creating Strings Using Pointers
- 11.6 Common Programming and Compiler Errors

11.1 Array Names as Pointers

- The address of an array
  - The **starting address** of an array is called the **base address of the array**.
  - **数组的基地址就是首元素地址**
  - The address of each successive element of the array is **offset** from the base by the size of the array type.

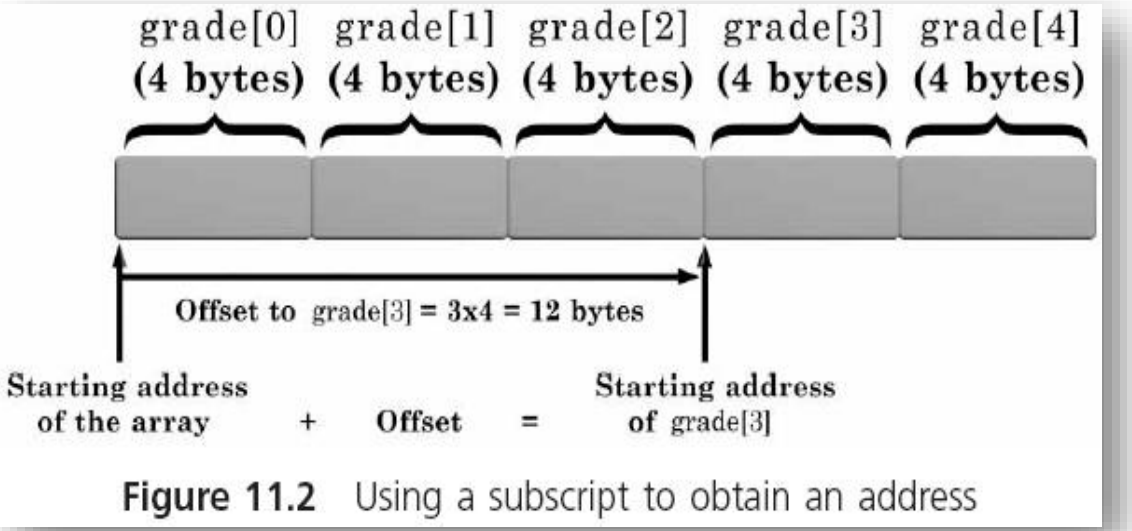

Figure 11.2 Using a subscript to obtain an address

Name	Value	Type
&grade[0]	0x001ffbc4	int *
&grade[1]	0x001ffbc8	int *
&grade[2]	0x001ffbcc	int *
&grade[3]	0x001ffbd0	int *
&grade[4]	0x001ffbd4	int *

11.1 Array Names as Pointers

- Array Elements May be Accessed in Two Ways

Table 11.1 Array Elements May be Accessed in Two Ways

Array Element	Subscript Notation	Pointer Notation
Element 0	`grade[0]`	`*gPtr`
Element 1	`grade[1]`	`*(gPtr + 1)`
Element 2	`grade[2]`	`*(gPtr + 2)`
Element 3	`grade[3]`	`*(gPtr + 3)`
Element 4	`grade[4]`	`*(gPtr + 4)`

gPtr为数组首元素指针

gPtr+1 为下一个元素的指针

11.1 Array Names as Pointers

- Program 11.1 Subscript Notation 下标法引用数组元素

```
#include <stdio.h>
#define NUMELS 5
int main()
{
  int grade[] = {98, 87, 92, 79, 85};
  for (int i = 0; i < NUMELS; i++)
    printf("Element %d is %d\n", i, grade[i]);
  return 0;
}
```

11.1 Array Names as Pointers

➢ Program 11.2 指针法**引用数组元素**

```
#include <stdio.h>
#define NUMELS 5
int main(){
        int *gPtr; /* declare a pointer to an int */
        int grade[] = {98, 87, 92, 79, 85};
        /* store the starting array address */
        gPtr = &grade[0];//gPtr = grade
        for (int i = 0; i < NUMELS; i++)
                printf("Element %d is %d\n",i,*(gPtr+ i));
        return 0;
}
```

gPtr为数组首元素指针

gPtr+i 为第i个元素的指针

11.1 Array Names as Pointers

- In C, adding 1 to a pointer makes the resulting ***pointer point to the next Element of the array***.

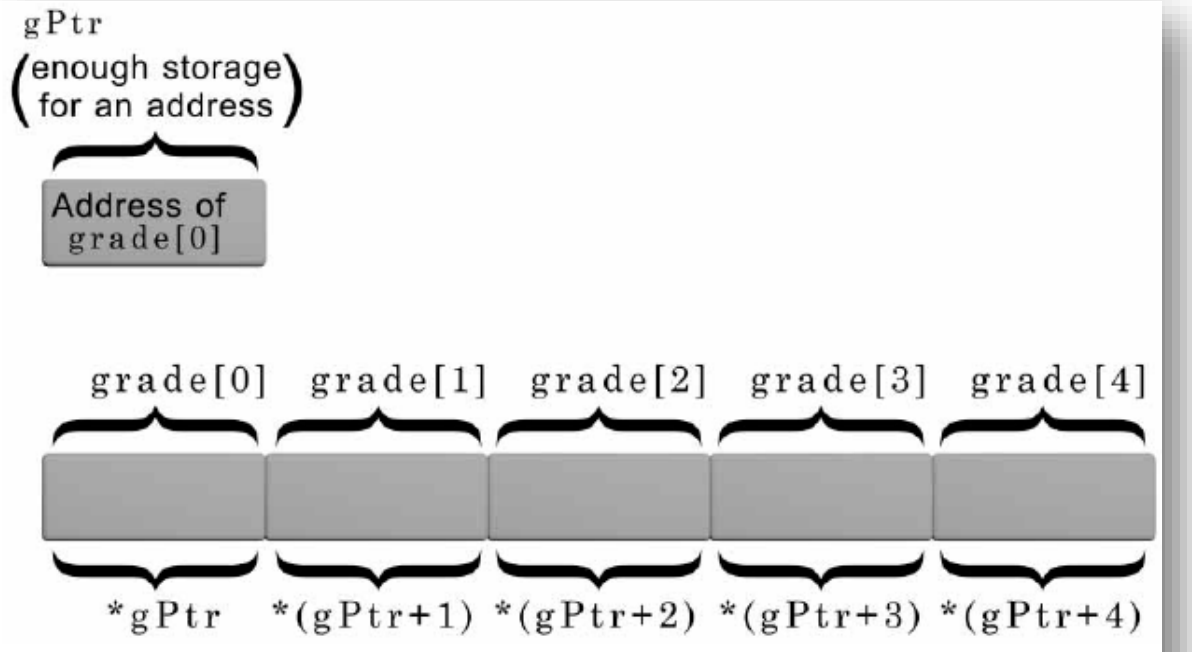

Figure 11.5 The relationship between array elements and pointers

Name	Value	Type
&grade[0]	0x001efb84	int *
&grade[1]	0x001efb88	int *
&grade[2]	0x001efb8c	int *
&grade[3]	0x001efb90	int *
&grade[4]	0x001efb94	int *
gPtr	0x001efb84	int *
gPtr+1	0x001efb88	int *
gPtr+2	0x001efb8c	int *
gPtr+3	0x001efb90	int *
gPtr+4	0x001efb94	int *

11.1 Array Names as Pointers

➢ Array Name as Pointer constant

– When an array is created, the compiler automatically creates an internal ***pointer constant*** (Array Name) for it and stores the base address of the array in this pointer

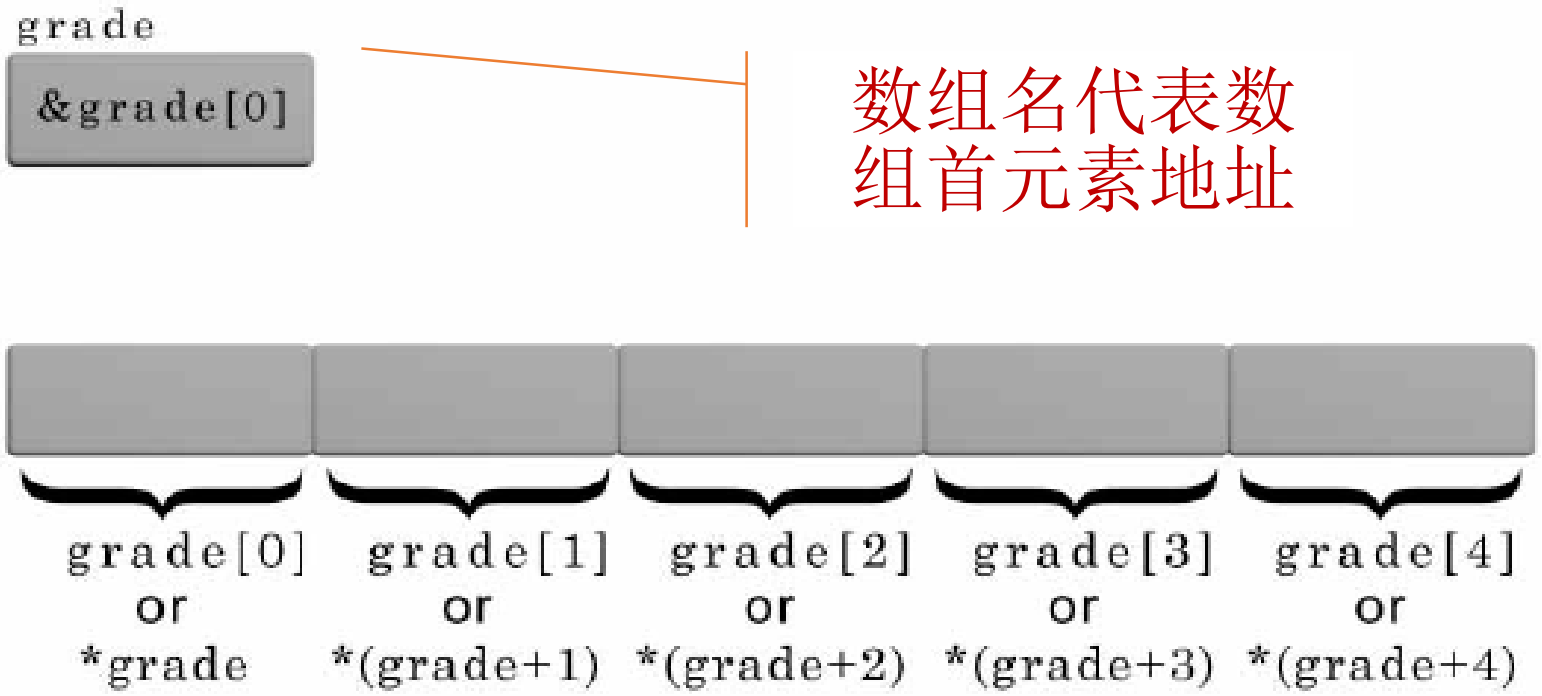

Figure 11.6 Creating an array also creates a pointer

11.1 Array Names as Pointers

- Program 11.3 Array Name as **Pointer constant**

```
#include <stdio.h>
#define NUMELS 5
int main()
{
        int grade[] = {98, 87, 92, 79, 85};
        for (int i = 0; i < NUMELS; i++)
        printf("Element %d is %d\n", i , *(grade + i) );
        return 0;
}
```

地址法**引用数组元素**

11.1 Array Names as Pointers

- Array Name as Pointer constant
  - In most respects **an array name** and **a pointer** can be used interchangeably
  - An array name is **a pointer constant**
  - grade = &grade[2]; //is invalid
  - **A pointer access can always be replaced using subscript notation**
  - **numPtr[i]** is valid even if numPtr is a pointer variable

11.2 Manipulating Pointers

- Manipulating Pointers
  - A pointer, constructed either as a variable or function parameter, contains a value: **an address**

  ① By **adding** numbers to and **subtracting** numbers from pointers, we can obtain different addresses

  ② The addresses in pointers can be compared using any of the **relational operators**(==, !=, <, >, etc.)

  ③ Pointers can be **initialized** when they are declared

11.2 Manipulating Pointers

- Pointer Arithmetic 指针算术运算
  - adding 1 to a pointer makes the resulting pointer point to the next Element of the array

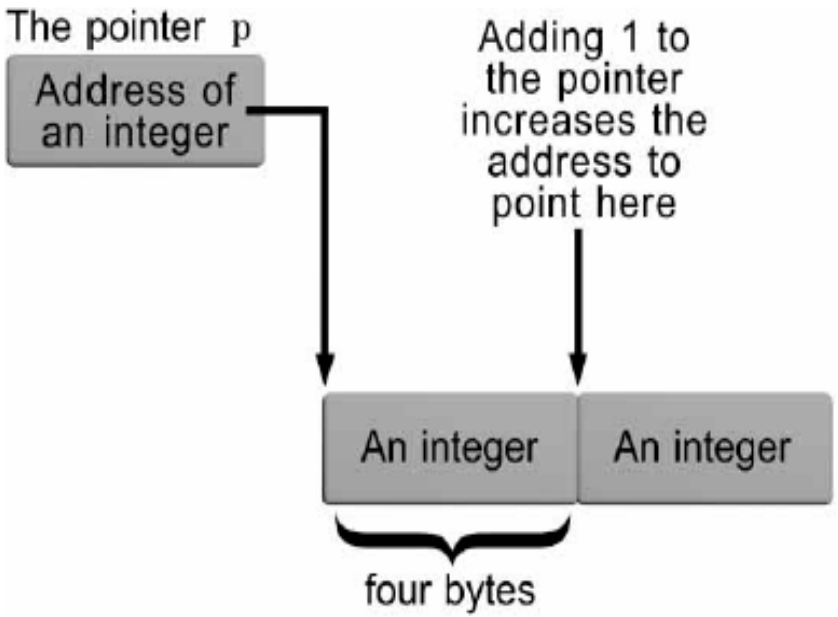

Figure 11.8 Increments are scaled when used with pointers

Name	Value	Type
&grade[0]	0x001efb84	int *
&grade[1]	0x001efb88	int *
&grade[2]	0x001efb8c	int *
&grade[3]	0x001efb90	int *
&grade[4]	0x001efb94	int *
gPtr	0x001efb84	int *
gPtr+1	0x001efb88	int *
gPtr+2	0x001efb8c	int *
gPtr+3	0x001efb90	int *
gPtr+4	0x001efb94	int *

11.2 Manipulating Pointers

➢ Program 11.4 Pointer Arithmetic

```
#include <stdio.h>
#define NUMELS 5
int main()
{
        int nums[NUMELS] = {16, 54, 7, 43, -5};
        int i, total = 0, *nPtr;
        nPtr = nums; //store address of nums[0] in nPtr
        for (i = 0; i < NUMELS; i++)
        total = total + *nPtr++;
        printf("The total of the array elements is %d\n", total);
        return 0;
}
```

11.2 Manipulating Pointers

- Pointer Initialization 指针初始化
  - Pointers can be initialized when they are declared:
  - **int** *ptNum = &miles;
  - **double** *zing = &prices[0];
  - **double** *zing = prices;

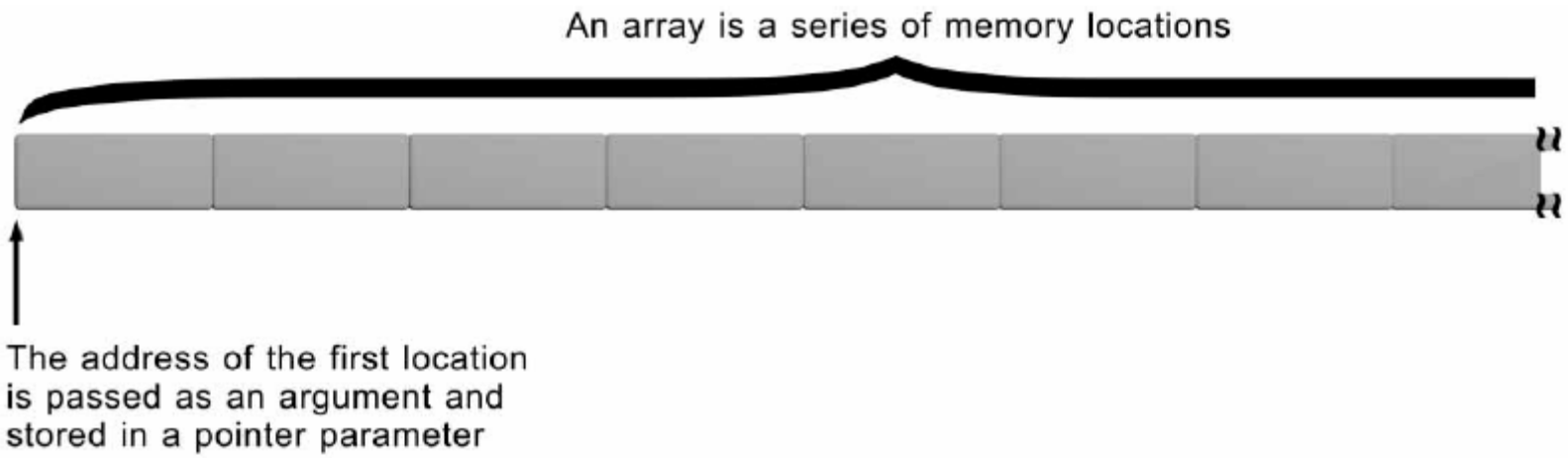

Figure 11.9 The address of an array is the address of the first location reserved for the array

11.3 Passing and Using Array Addresses

- Program 11.6 Passing and Using Array Addresses

```
#include <stdio.h>
#define NUMELS 5
int findMax(int[], int);
int main()
{
        int nums[NUMELS]={2, 18, 1, 27, 16};
        printf("The maximum value is %d\n", findMax(nums, NUMELS));
        return 0;
}
```

Calling findMax(&nums[2], 3) would be valid too

11.3 Passing and Using Array Addresses

➢ Program 11.6 Passing and Using Array Addresses

```
11 int findMax(int vals[], int numEls)
12 {
13     int i, max = vals[0];
14     for (i = 1; i < numEls; i++)
15         if (max < vals[i])
16             max = vals[i];
17     return (max);
18 }
```

findMax 0x00d61490 findMax(int *, int)

Can be replaced with
findMax(int *vals, int numEls)

Note: **vals** is a pointer parameter; thus, its address can be modified (but **nums**' address in main(), cannot).

11.3 Passing and Using Array Addresses

- findMax() can be rewritten as:

```
int findMax(int *vals, int numEls)
{
    int max = *vals++;
    for (int i = 1; i < numEls; i++,vals++)
    {
        if (max < *vals)
            max = *vals;
    }
    return (max);
}
```

11.3 Passing and Using Array Addresses

- Advanced Pointer Notation
  - #define ROWS 2
  - #define COLS 3
  - int nums[ROWS][COLS] = { {16,18,20},
  - {25,26,27} };

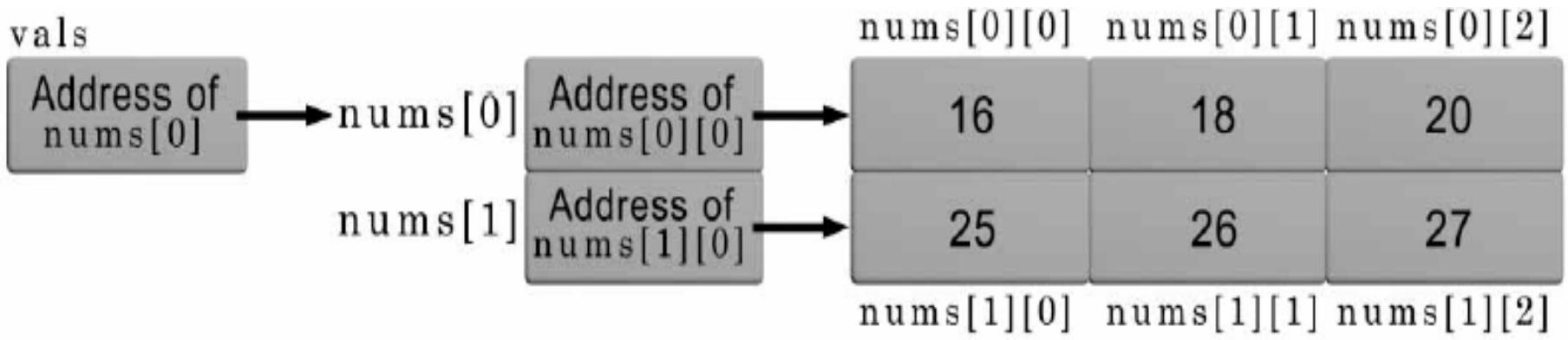

Figure 11.11 Storage of the nums array and associated pointer constants

11.3 Passing and Using Array Addresses

- Advanced Pointer Notation

Name	Value	Type
*(*nums)	16	int
*(*nums+1)	18	int
*(*nums+2)	20	int
((nums+1))	25	int
((nums+1)+1)	26	int
((nums+1)+2)	27	int

Name	Value	Type
nums[0][0]	16	int
nums[0][1]	18	int
nums[0][2]	20	int
nums[1][0]	25	int
nums[1][1]	26	int
nums[1][2]	27	int

Name	Value	Type
*nums[0]	16	int
*(nums[0]+1)	18	int
*(nums[0]+2)	20	int
*nums[1]	25	int
*(nums[1]+1)	26	int
*(nums[1]+2)	27	int

Name	Value	Type
nums[0][0]	16	int
nums[0][1]	18	int
nums[0][2]	20	int
nums[1][0]	25	int
nums[1][1]	26	int
nums[1][2]	27	int

11.3 Passing and Using Array Addresses

- Advanced Pointer Notation
  - A function that receives an integer two-dimensional array can be declared as:
  - calc(**int pt[2][3]**)
  - calc(**int pt[][3]**)
  - calc(**int (*pt)[3]**)
  - It refers to a single pointer of objects of three integers

11.3 Passing and Using Array Addresses

- Advanced Pointer Notation
  - Once the correct declaration for ***pt*** is made (any of the three valid declarations), the following notations within the function calc() are all equivalent:

Pointer Notation	Subscript Notation	Value
`*(*pt)`	`pt[0][0]`	16
`*(*pt+1)`	`pt[0][1]`	18
`*(*pt+2)`	`pt[0][2]`	20
`*(*(pt+1))`	`pt[1][0]`	25
`*(*(pt+1)+1)`	`pt[1][1]`	26
`*(*(pt+1)+2)`	`pt[1][2]`	27

11.3 Passing and Using Array Addresses

- Advanced Pointer Notation
  - A function can return a pointer
    - int *calc()
  - Pointers to functions are possible because function names, like array names, are themselves **pointer constants**
    - int (*calc)()
    - Declares **calc** to be a pointer to a function that returns an integer
    - If, for example, **sum()** returns an integer, the assignment **calc** = **sum**; is valid

11.4 Processing Strings Using Pointers

I. Processing Strings Using Array

```
void strcopy(char string1[], char string2[])
{
        int i = 0;
        while (string1[i] = string2[i++]);
}
```

II. Processing Strings Using Pointers

```
void strcopy(char *string1, char *string2)
{
        while (*string1++ = *string2++);
}
```

11.5 Creating Strings Using Pointers

- **A pointer constant**
  - char **message** [81] ="this is a string";
  - char *message1=message;
- **Create a string using a pointer**
  - char *message2="this is a string";
  - message2++;

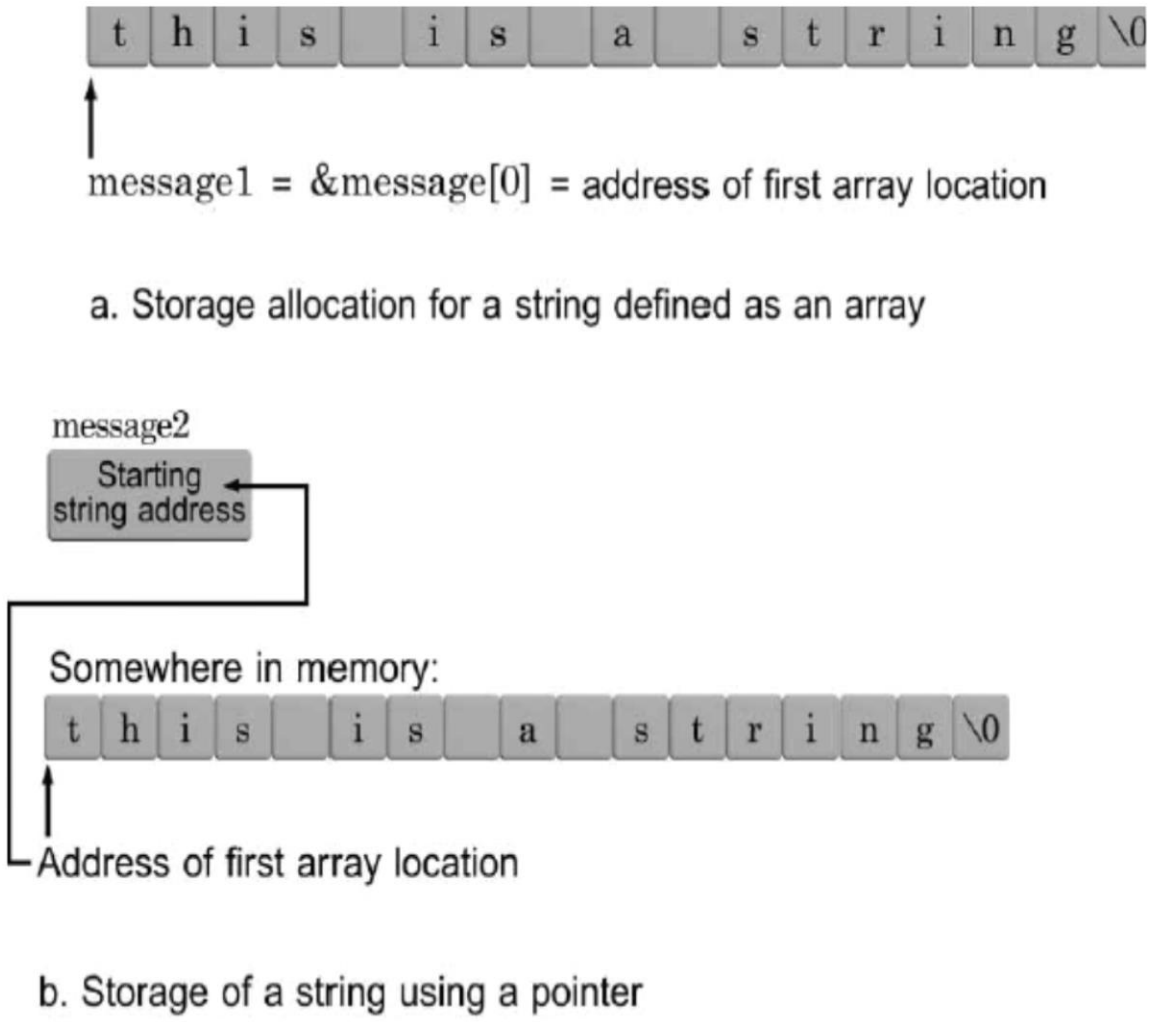

Figure 11.12 String storage allocation

11.5 Creating Strings Using Pointers

- Program 11.7 character pointer as String;

Name	Value	Type
message2	0x012c5758 "A new messag	char *

```
#include <stdio.h>
int main()
{
        char *message2 = "this is a string";
        printf("\nThe string is %s", message2);
        printf("\n The base address of this string is
        %p\n", message2);
        message2 = "A new message";
        printf("\nThe string is now: %s", message2);
        printf("\n The base address of this string is
        %p\n", message2);
        return 0;
}
```

11.5 Creating Strings Using Pointers

- **Allocating Space for a String**
  - The following declaration is valid:
    - **char** *message = "abcdef"; **//valid**
  - But, this is not:
    - **char** *message;
    - strcpy(message,"abcdef"); **//INVALID**

11.5 Creating Strings Using Pointers

- Pointer Arrays
  - Example:
    - **char** *seasons[4]; seasons[0] = "Winter";
    - seasons[1] = "Spring"; seasons[2] = "Summer";
    - seasons[3] = "Fall";
  - Or:
    - **char** *seasons[4]={"Winter", "Spring", "Summer", "Fall"};

Name	Value	Type
seasons	0x00f5f740	char * [4]
[0]	0x0026579c "Winter"	char *
[1]	0x00265768 "Spring"	char *
[2]	0x00265758 "Summer"	char *
[3]	0x0026573c "Fall"	char *

11.5 Creating Strings Using Pointers

➢ Program 11.8 Pointer Arrays

```
#include <stdio.h>
int main()
{
        int n;
        char *seasons[] = {"Winter", "Spring", "Summer", "Fall"};
        for(n = 0; n < 4; n++)
        printf("The season is %s.\n", seasons[n]);
        return 0;
}
```

11.6 Summary

- An array name is a pointer constant
- Any access to an array element using subscript notation can always be replaced using pointer notation
- Arrays are passed to functions by address, not by value
- When a single-dimensional array is passed to a function, the parameter declaration for the array can be either an array declaration or a pointer declaration
- In place of subscripts, pointer notation and pointer arithmetic are especially useful for manipulating string elements
- String storage can be created by declaring an array of characters or a pointer to be a character
- Pointers can be incremented, decremented, and compared

Reference

- https://www.codesdope.com/blog/article/int-main-vs-void-main-vs-int-mainvoid-in-c-c/