C Arrays

A series of variables ordered by index and treated as a single block is called indexed variables. These variables can be of the same type to take the form of an **array**. An array, that contains character data-type variables, is known as a **string**. Indexed variables with different data type are called a **structure**. This chapter has extensively covered two types of indexed variables array and string. The structure it will be discussed in the next chapter [2 p.36].

1. Objectives of the chapter

- Recall the syntax for declaring arrays in C.
- Remember the concept of array indexing and its starting index in C (0 or 1).
- Understand how arrays are stored in memory and how elements are accessed using pointers.
- Write C code to initialize arrays with predefined values.
- Evaluate the performance of algorithms that use arrays for tasks such as searching or sorting large datasets.

2. Required tribal gains

1. First, we should know the reason to use array in C. Why we need array in C?
2. **Memory Management**: Understand the basics of memory management in C, including stack vs. heap memory, memory allocation/deallocation functions (malloc, calloc, realloc, free), and the concept of memory leaks
3. **Array Declaration**: How to declare arrays in C.

3. Test of tribal gains

Quiz 1

[solution n°4 p. 33]

What is an array in C?

- ☐ A single variable that can hold multiple values of different data types
- ☐ A collection of elements of the same data type stored under a single identifier
- ☐ A reserved keyword used to define functions

Quiz 2

[solution n°5 p. 33]

How do you declare an array in C?

- ☐ int array[5];
- ☐ array = int[5];

Quiz 3

[solution n°6 p. 34]

What is the index of the first element in an array in C?

- ☐ 0
- ☐ 1

Quiz 4

[solution n°7 p. 34]

How do you access the third element of an array named `numbers` in C?

- ☐ numbers(3);
- ☐ numbers{2};
- ☐ numbers[2];

4. Definition

Definition

- An array is a method to treat an ensemble of multiple variables with the same data-type as a one class.
- Each variable in the array represents an **element** in this array.
- Each variable occupy a storage space in memory, and the storage space of all of the array elements is called ***contiguous memory***[5 p.36] allocation.
- Elements in array are ordered by **index** in which the 0th index refers to the first element in the array.

5. Array declaration

Declaring an array is done in the following way

data_type array_name[array_size];

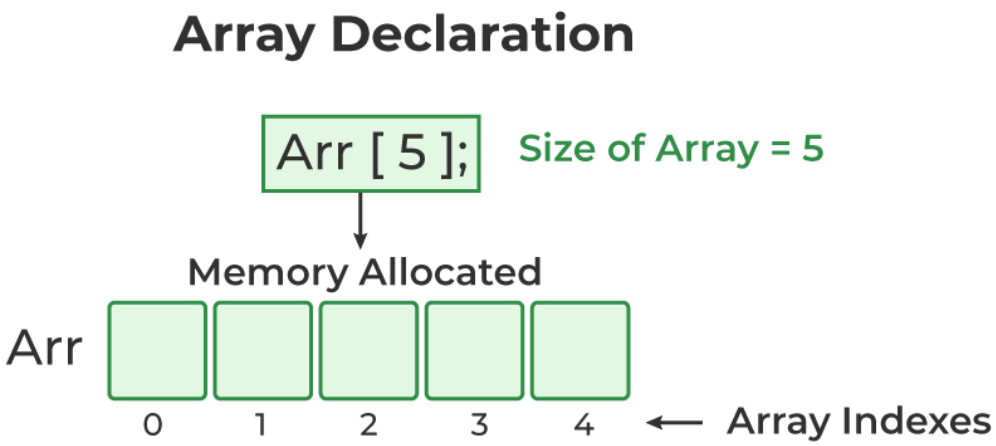
- **Data-type** : is defined according to the data-type of the array elements.
- **Array-name**: is chosen by the programmer and it is important and used to refer to the array element.
- **Array size**: is the number of elements in the declared array.

Array declaration

Example

int arry[6];

int is the data type of the six elements in this array, and this statement can be pictured as follows



Vusialisation of array elements allocated in memory

Note

It is very important to note the following:

- The array element is referred to by its name followed by an index placed between square brackets.
- The index of first element of the array is 0, and therefore always there is a difference of one between the index of the last element and the size of the array.

6. Array initialization

The method of assigning an initial value to a variable is known as initialization. There are many ways to initialize an array.

6.1. Initialization during declaration step

Syntax:

data_type array_name[array_size]={element 1, element 1, element 1,...};

Example

int x[6]={2, 4, 6, 8, 5, 1};

Note

In this case, it is not necessary to specify the size of the array and the initialization remains correct in this way

int x[ ]={2, 4, 6, 8, 5, 1};

6.2. Initialization using loops

Example

Program	Output
#include <stdio.h> int main() { int i,x[6]; for(i=0;i<6;i++) { printf("Enter the element x[%d]:\n", i); scanf("%d", &x[i]); } return 0; }	Enter the element x[0]: 2 Enter the element x[1]: 4 Enter the element x[2]: 6 Enter the element x[3]: 8 Enter the element x[4]: 5 Enter the element x[5]: 1

7. Access array elements

Array elements are manipulated as variables, and index is the only difference between elements and variables. The index requires an arrangement in memory to distinguish these elements from each other. Therefore, the element can only be accessed through this index.

Access, here, means two things:

- Using the value of the element
- Dealing with the memory address of the element.

array_name[0] accesses the first element in the array, while array_name[1] accesses the second element, etc.

Example

Program	Output
#include <stdio.h> int main() { int i,a,b=2, y[5]={5,b,1,4,6}; a=y[0]; printf("a=%d \n %d",a, y[1]); return 0; }	a=5 2

An opportunity presents itself here to demonstrate a straightforward method for assigning values to elements of an array.

int arr[3] ;

arr[0]=5 ;

arr[1]=12 ;

arr[2]=16 ;

8. Input/output an element of array

- To input the first and second elements of the array float arr[n], we just use scanf() function as follows

scanf("%f%f", &arr[0], &arr[1]) ;

- To output the first and the second elements of the last array we just use printf() function in this way

  printf("%f%f", arr[0], arr[1]) ;

9. Pointers in C

9.1. C Memory address

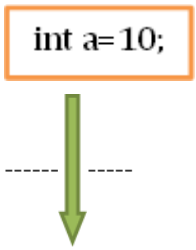
Definition

C memory address is the space required for storing a declared variable, or more appropriately, it is the location of bytes occupied by the variable data (value) in the computer.

For example the statement int a=10 ; leads to :

1- A storage size with 4 bytes is reserved in the memory.

2- The value 10 of a is converted to binary as (10)10 =1010 and stored in the reserved memory of a as follows



Address of a	Data of a=10								
1992					1	0	1	0	1byte=8bits
1993									
1994									
1995									

Data storage of a

In a C program, the address memory is frequently presented, particularly when using the `scanf()` function.

The statement `scanf(%d, &b);` means assigning a value to the address of b, which is referred to as &b.

It should be noted that the address operator *"&* "can be used to access the memory address of a variable.

9.2. Pointers in C

Definition

Pointers are variables that store the address of another variable as their values. In c, pointers it can be created as follows :

- Declaration : Pointer is declared in this way

  `pointer_data-type* pointer_name ;`

  The only difference between pointer and variable declarations is the astrisic (*).

  Example :

  `int* p ;`

  This line means that a pointer is created but it points to a random adress memory. The declared pointer keeps pointing to an existing and unknown address in memory, and if it is not specified, a programming issue may arise. To this reason, a care should be done in pointers manipulation.

- Initialization: the pointer is initialized by using address operator '&' as

  int a=5, *pa ;

  pa=&a ;

  This indicates that the pointer pa has been assigned the address of a. Also, the stored value of a is accessed using the dereference operator *.

Example

The following example will clarify everything that has been mentioned in this section.

Program	Output
#include <stdio.h> int main() { int a=5, *pa;* *pa=&a;* `printf("The value of a=%d\n",a); printf("The address of a: %p\n", &a); printf("The value of a=%d\n", pa); printf("The address of a: %p", pa); return 0; }`	The value of a=5 The address of a= 0x7ffe665a0c14 The value of a=5 The address of a= 0x7ffe665a0c14

From this example, the importance of pointers also becomes clear, as they play two roles. On the one hand, they store the address of the variable, and on the other hand, they can access the value of the variable they pointed to. Even beyond that, the value of the variable it can be changed through its pointer.

Example

Program	*Output*
#include <stdio.h> int main() { int a=5, *pa; pa=&a; printf("The value of a=%d\n",a); printf("The address of a: %p\n", &a); printf("The value of a=%d\n", *pa); printf("The address of a: %p", pa); return 0; }	The value of a=5 The address of a= 0x7ffe665a0c14 The value of a=5 The address of a= 0x7ffe665a0c14

9.3. Pointers and functions:

Pointers are useful tools to deal with functions. It can be pointed to the function itself or it can be used as a parameter to that function. In most cases pointers are used to pass addresses to function parameters and this technique is called pass by reference. This it can be explored in the programs as follows

Example

Program	*Output*
#include <stdio.h> void fg(int *q) { printf("%p\n%d", q,*q); } int main() { int a=5, *p; p=&a; printf("%p\n",p); fg(p); return 0; }	0x7ffd26425cf4 0x7ffd26425cf4 5

Example

Program	*Output*
#include <stdio.h> void swap(int *p1, int *p2) { int temp; temp=*p1; *p1=*p2; *p2=temp; printf("x=%d\ny=%d\n",*p1,*p2); } int main() { int x=5,y=10; printf("The value of x and y before swap\n"); printf("x=%d\ny=%d\n", x, y); printf("The value of x and y after swap\n"); swap(&x,&y); return 0; }	The value of x and y before swap x=5 y=10 The value of x and y after swap x=10 y=5

9.4. Pointers and arrays

The pointer of an array can be shown as a second array, and the elements of this array store the addresses of the first array elements to which it points.

Example

Program	*Output*
#include <stdio.h> int main() { int i, j, a[4]={5,4,3,1}; int* pa; pa=&a; printf("The address of a[] : %p\n\n", pa); printf("The addresses of array's elements:\n");	The address of a[] : 0x7ffe14beea20 The addresses of array's elements: a[0] : 0x7ffe14beea20 a[1] : 0x7ffe14beea24 a[2] : 0x7ffe14beea28

for(i=0;i<4;i++) { printf("a[%d] : %p\n", i, pa+i); } printf("The values of array's elements:\n"); for(j=0; j<4; j++) { printf("a[%d]=%d\n", j, *(pa+j)); } return 0; }	a[3] : 0x7ffe14beea2c The values of array's elements: a[0]=5 a[1]=4 a[2]=3 a[3]=1

From the previous example, it can be noted that:

- The address of array a[] corresponds to the address of a[0].
- The addresses are shifted by 4, and this is due to the fact that each element with data-type integer reserves 4 bytes in memory.
- The method to access array elements is done in the same way as

a[0]=*pa=5,

a[1]=*(pa+1)=4,

a[2]=*(pa+2)=3,

a[3]=*(pa+3)=1

Pointers can only be used for one element in the array.

int c[]={1, 2, 3}, *pc;

pc=&c[1];

In this case the order becomes important, so pc+1 refers to the second element after c[1] and pc-1 refers to the second element before c[1].

? *Example*

Program	*Output*
#include <stdio.h> int main() { int c[]={1, 2, 3}, *pc; pc=&c[1]; printf("%d\n%d", *(pc-1), *(pc+1)); }	1 3

Example

Here is a simple example of printing elements for every given array using a pointer, function, and an array.

Program	*Output*
##include <stdio.h> void funcar(int *pa, int size) { int i; for(i=1; i<size; i++) { printf("a[%d]=%d\n", i, *(pa+i)); } } int main() { int b, a[]={4, 2, 3, 12, 51}, *p; p=&a; b=sizeof(a)/sizeof(a[0]); funcar(p,b); return 0; }	a[1]=2 a[2]=3 a[3]=12 a[4]=51

In this chapter, we have delved into the intricate world of C pointers, one of the most powerful and fundamental concepts in the C programming language. We began by understanding the basics of pointers, including their declaration, initialization, and dereferencing. We then explored how pointers can be used to manipulate memory directly.

10. Two dimensional array

10.1. Definition

Definition

The two dimensional array (2D array) it can be viewed as an array of arrays. The 2D array is required in the case of stored data that takes *tabular form or matrices*[6 p.36].

10.2. Declaration

Declaration of 2D array can be done by the following task

```
data type array_name[x][y];
```

Example:

long M[2][3];

This statement can be represented as

	col 0	col 1	col 2
row 0	M[0][0]	M[0][1]	M[0][2]
row 1	M[1][0]	M[1][1]	M[1][2]

long: is the data type of the 2D array.

M: is the name of the 2D array.

2: between square brackets represents the number of rows in the 2D array.

3: is the number of columns in the 2D array.

10.3. Initializing 2D array

There are many ways to initialize a 2D array the simple one is the initialization during declaration as

float X[3][3]={1.2, 5, 10, 2, 4, 3, 6, 9, 7};

Or in equivalent way as

float X[3][3]={{1.2, 5, 10}, {2, 4, 3}, {6, 9, 7}};

The two statements can be visualized as follows

	col 0	col 1	col 2
row 0	X[0][0]=1.2	X[0][1]=5	X[0][2]=10
row 1	X[1][0]=2	X[1][1]=4	X[1][2]=3
row 2	X[2][0]=6	X[2][1]=9	X[2][2]=7

To assign values to elements in 2D array the scanf() can be used as follows

int M[2][2];

scanf("%d", &M[0][0]);/* to enter the value of the first element in array M*/

10.4. Printing 2D arrays:

To print all elements of 2D array nested for loop is used as follows:

Program	*Output*
#include <stdio.h> int main() { int M[3][4]= {1,2,3,7,2,4,5,8,3,5,6,9}; int i, j; for(i=0;i<3;i++) {	1 2 3 7 2 4 5 8 3 5 6 9

```
    for(j=0;j<4;j++)
    {
       printf("%d ", M[i][j]);
    }
    printf("\n");
  }


  return 0;
}
```

10.5. Storage of 2D array

The elements of 2D array are stored in memory in a linear way in row-major order i.e. the rows are placed one by one in memory as one dimensional array. The element of the 2D array int x[3][2]={1, 2, 2, 1, 1, 4}; it can be represented in this way

Element	**x[0][0]**	**x[0][1]**	**x[1][0]**	**x[1][1]**	**x[2][0]**	**x[2][1]**
Value	1	2	2	1	1	4
Address	2000	2004	2008	2012	2016	2020
	row 0		row 1		row 2	

Example

Program	*Output*
#include <stdio.h> int main() { int mat[3][2]={{1,2},{5,9},{7,9}}; int i,j; int* pt; pt=&mat[0][0]; for(i=0;i<6;i++) printf("%p\n", pt+i); return 0; }	0x7ffe872ca670 0x7ffe872ca674 0x7ffe872ca678 0x7ffe872ca67c 0x7ffe872ca680 0x7ffe872ca684

10.6. Length of 2D array

The total number of elements in 2D array=number of rows ´ number of columns. For example the 2D array int A[10][7] can store 10´7=70 elements. Each of element occupy a space of 4 bytes in memory, and then the array has a storage space given with

size of A[10][7] =70´4=280 bytes.

Example

Program	*Output*
#include <stdio.h> int main() { int mat[3][2]={{1,2},{5,9},{7,9}}; int length; length=sizeof(mat)/sizeof(mat[0][0]); printf("length=%d", length); return 0; }	length=6

10.7. Some operations on 2D arrays

C program to find 0^{th} column and 0^{th} row of 2D array:

Example

Program	*Output*
#include <stdio.h> int main() { int a[3][3]={{4,3,2},{5,4,1},{6,9,8}}; int i, j; printf("elements of 0-column:\n"); for(i=0;i<3;i++) { printf("%d\n", a[i][0]); } printf("\n----------\n"); printf("elements of 0-row:\n"); for(j=0;j<3;j++) { printf("%d\t", a[0][j]); } return 0; }	elements of 0-column: 4 5 6 ---------- elements of 0-row: 4 3 2

Program to find the position of an element in 2D array *Example*

Program	*Output*
#include <stdio.h> int main() { int b[4][4]={{6,4,3,2},{10,5,4,1},{4,6,9,8},{9,8,7,12}}; int i, j, a, c=0; printf("Enter the number you are looking for:\n"); scanf("%d", &a); for(i=0;i<3;i++) { for(j=0;j<3;j++) { if(a==b[i][j]) { printf("The number position is (%d, %d)\n", i, j); c++; } } } if(c==0) { printf("Not found"); } return 0; }	Enter the number you are looking for: 4 The number position is (0, 1) The number position is (1, 2) The number position is (2, 0)

C program to solve a system of equations with two variables using matrices: *Example*

a-The form of a system of equation can be written as

$a_1x + b_1y = c_1 a_2x + b_2y = c_2$

The theoretical solution of this system of equations is given by

$$x = \frac{\begin{vmatrix} c1 & b1 \\ c2 & b2 \end{vmatrix}}{\begin{vmatrix} a1 & b1 \\ a2 & b2 \end{vmatrix}}$$

$$y = \frac{\begin{vmatrix} a1 & c1 \\ a2 & c2 \end{vmatrix}}{\begin{vmatrix} a1 & b1 \\ a2 & b2 \end{vmatrix}}$$

Where $\begin{vmatrix} a1 & b1 \\ a2 & b2 \end{vmatrix} = a_1b_2 - a_2b_1$ is the determinant of the matrix.

b- Next, we will write the C program that finds a solution of the following system of equations

$2x + 4y = 1x + 3y = 2$

Program	*Output*
#include <stdio.h> float determinanat(float a[2][2]) { float det; det=a[0][0]*a[1][1]-a[1][0]*a[0][1]; return det;} int main(){ float b[2][2]; float b1[2][2]; float b2[2][2]; int i,j,k,l; printf("Enter the elements of the matrix\n"); for(i=0;i<2;i++) { for(j=0;j<2;j++) { scanf("%f",&b[i][j]); } } for(k=0;k<2;k++) { for(l=0;l<2;l++) { b1[k][l]=b[k][l]; b2[k][l]=b[k][l]; }} float c1=1, c2=2, x, y, d, z1, z2; d=determinanat(b); printf("d=%f\n", d); b1[0][0]=c1;	Enter the elements of the matrix 2 4 1 3 d=2.000000 x=-2.500000 y=1.500000 solutions substitutions z1=0.000000 z2=0.000000

```
b1[1][0]=c2;
b2[0][1]=c1;
b2[1][1]=c2;
x=determinanat(b1)/d;
y=determinanat(b2)/d;
printf("x=%f\ny=%f\n", x, y);
printf("solutions substitutions\n");
z1=2*x+4*y-1;
z2=x+3*y-2;
printf("z1=%f\nz2=%f", z1, z2);
return 0;}
```

In conclusion, arrays are fundamental data structures in the C programming language that allow us to store multiple elements of the same data type under a single identifier. Throughout this chapter, we've explored the various aspects of arrays, starting from their declaration and initialization to accessing elements and performing operations on them.

Series of exercises

You will find in the link some exercises related to the subject

https://drive.google.com/file/d/1kVF9APhBRdzrr0qDhF_PDKeDdamNtk8h/view?usp=sharing