# Introduction to array

## Definition of array and its features

Q: [What is array and what are its features?](#array_definition)

Q: [Given an array of integers, how the system calculates address of its elements?](#example_array_of_ints)

Q: [Why elements of an array need to be of same type?](#why_same_type_elements)

Q: [Why elements of an array need to be put in consecutive spaces?](#why_put_in_consecutive_spaces)

Q: [Is the index of an array always start with zero?](#index_start_at_zero)

## Declare and initialization an array

Q: [How to declare an array in general?](#declare_an_array_in_general)

Q: [How to declare an array of ints?](#declare_an_array_of_ints)

Q: [How to declare an array of strings?](#declare_an_array_of_strings)

## Common mistakes

Q: [Is int arr[3]; the same as arr[3]?](#int_arr_3_same_as_arr_3)

Q: [What is a common mistake in index?](#common_mistake_in_index)

## Miscellaneous

Q: [Why array?](#why_array)

Q: [What are the pros and cons of array?](#pros_cons_array)

Q: [Are there other ways to organize a collection of elements?](#other_way_to_hold_collection_of_elements)

**Q: What is an array and what are its features?**

A: An array is a collection of same type elements residing in consecutive memory locations. It has three features:

1. **collection** of elements
2. elements are of the **same** type.
3. These same-type elements reside in **consecutive** memory locations

Here is an analog. Suppose a group of students live in a hotel.

1. Each student resides in one room.
2. The group reside in consecutive rooms, that is, the rooms are adjacent to each other.

Thus, if we know the room number of the first student and the size of the group, then we can find each student through their room numbers. For example, if the first student lives in Room 200 and the number of student is 5, then the second student must live in Room 201 and the last student must live in Room 204. With the room number, we can visit the student. In the view of computer, with an address, we can access the element in that address.

**Q: Given an array of integers, how the system calculates address of its elements?**

A: Here is an example.

* Suppose we have 5 integers 1, 2, 3, 4, 5 in an array.
* Note that each integer in C++ takes 4 bytes.
* Suppose the initial address of the array (also called base address of array) is 1000, which is decided during compilation or run time, when system finds a not-occupied-yet memory block big enough to hold 5 integers consecutively. Programmers do not need to worry about the actual address.

element index in the array address

1. 0 1000 = 1000 + 0 \* 4
2. 1 1004 = 1000 + 1 \* 4
3. 2 1008 = 1000 + 2 \* 4
4. 3 1012 = 1000 + 3 \* 4
5. 4 1016 = 1000 + 4 \* 4

* Address of the element at index i (index starts from zero) is calculated by

base address + i \* size\_of\_single\_element\_in\_that\_array

Once we know the address, we can access that element directly.

* Calculation of address takes constant time (one multiplication, one addition), so to access an individual element in the array is fast.

**Q: Why elements of an array need to be of same type?**

A: If elements in an array are not of the same type, then we cannot calculate the address of an element based only on its index and base address, where base address is the address of the first element in array. An illustration of calculation the address of element with a given index is shown in an [example of calculate addresses of elements in array](#example_array_of_ints).

**Q:** **Why elements of an array need to be put in consecutive spaces?**

A: Putting same type elements in consecutive spaces enables the access of an individual element in done in constant time. See an [example of calculate addresses of elements in array](#example_array_of_ints).

**Q: Is the index of an array always start with zero?**

A: Yes. The index of an array must start with zero; no exception to the rule. You can look at the formula to calculate the address of element based on initial address of array and index at an [example of calculate addresses of elements in array](#example_array_of_ints).

**Q: How to declare an array in general?**

A: Depending on the type of array element, we use either one of the following.

type\_of\_array\_element array\_name[] = {value1, value2, ..., value3};

or

const int SIZE = a\_positive\_integer;

type\_of\_array\_element array\_name[SIZE];

**Q: How to declare an array of integers?**

A: To declare an array of integers, we have several approaches.

1. Declare an array of ints and initialize it with **exactly** three elements 2, 1, 6.

int arr[] = {2, 1, 6};

Or

int arr[3] = {2, 1, 6};

1. Declare an array of three ints. The elements are initialized later. Initialization can be done by assignment operator = as follows, or by input from keyboard or file.

int arr[3]; //Declare an array of ints that can hold three elements.

//The elements of this array are yet to be initialized.

arr[0] = 1;

   arr[1] = 2;

   arr[2] = 3;

1. Declare an array with a certain capacity and initialize it with first several elements and leave the rest elements to be zero.

const int CAPACITY = 6;

       //CAPACITY of an array needs to be const

       //to work with every C++ version.

    int arr[CAPACITY] = {1, 2, 3};

//Variable i represents the index of an array.

    //i starts from 0 and

//is strictly smaller than the array's capacity.

//The following for-statement prints out 1 2 3 0 0 0

    for (int i = 0; i < CAPACITY; i++)

        cout << arr[i] << " ";

**Q: How to declare an array of strings?**

A: Declare an array of strings as greetings. For example,

const int SIZE = 3;

string greetings[SIZE] =

{"Hello", "How are you?", "How do you do?"};

for (int i = 0; i < SIZE; i++)

    cout << greetings[i] << endl;

**Q: Is int arr[3]; the same as arr[3]?**

A: No.

int arr[3]; is not the same as arr[3].

Statement int arr[3]; declares arr as an array of ints with three elements (see type int before arr[3] in this denotation). Like any other declaration, the above declaration of array can only write once in a method.

After properly declare and initialize array arr, denotation arr[i] represent the element of arr indexed at i, where i is an int representing index. Whatever type an array may be, its index is always an int and always starts from zero.

For example, given an array of size 3, then arr[0] is the first element in the array, arr[1] is the second element in the array, and arr[2] is the third (and also the last element) in the array of size 3.

Given an array arr of size 3, the smallest index is 0. The largest index is 2, which is one smaller than the size; this is because we start index from 0.

**Q: What is a common mistake in index?**

A: Suppose variable i represents the index of an array, then the index starts from zero and is strictly smaller than the size of an array. The following code has some strange integer following elements 1 2 3 0 0 0 since arr has only 6 elements, so the index can only be in [0, 5], and arr[CAPACITY] is not part of this array and is not properly initialized.

const int CAPACITY = 6;

       //CAPACITY of an array needs to be const

       //to work with every C++ version.

    int arr[CAPACITY] = {1, 2, 3};

    //Element arr[CAPACITY] is undefined and

//does not have fixed value.

    for (int i = 0; i <= CAPACITY; i++)

        cout << arr[i] << " ";

Fix: change i <= CAPACITY to i < CAPACITY.

**Q:** **Why array?**

A: In some situations, we need to work with a group of elements of the same type, for example, to record grades of students in a class of size 300. If we had to use a variable to represent a grade, then we need 300 variables, that is awkward and will not work.

As another example, in Lab 3D at <https://tong-yee.github.io/135/labs/lab_03.html>, which reports the **West basin elevation** for all days in the interval in the reverse chronological order (from the later date to the earlier). Each value needs to be saved, and we do not how many values in advance. The duration can range from 1 day to 365 days. It is impractical to declare 365 variables.

**Q: What are the pros and cons of array?**

A: By put a group of same type elements in consecutive spaces, we have pros and cons.

Pros:

* Simple.
* An individual element can be accessed in constant time.

Cons:

* Insertion or deletion of an element may need to involving moving other elements. For example, if we delete the first element from array, we need to move the rest elements one space towards the head.
* Need to declare the size of array in the beginning, afterwards, you cannot change the capacity easily. Suppose in the beginning, we declare an array of size 100. Later on, you would like to expand the array to size 200. Note that elements of an array need to be located consecutively, however, there might not be enough free memory after the last element of the original array.

**Q: Are there other ways to organize a collection of elements?**

A: Yes. For example, linked list does not require to put elements in consecutive spaces, but then you need additional address to remember the address of next neighbor.

Another way is vector.