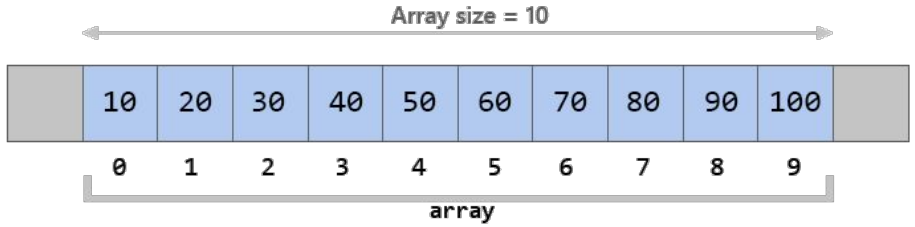
C Programming

Arrays and Strings

Arrays and Strings

- An array is the simplest data structure which contains elements, stored consecutively in a continuous piece of memory.
- The elements in an array have the same data type and it can be any object type



int array[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}

- An array can be defined by its identifier, type of the elements, and number of the elements in the array; i.e. **type name[number_of_elements];**
  - The number of elements shall be an integer greater than zero
- To get size (occupied memory) of an array, use the sizeof operator. E.g. *sizeof(array);*
- To get number of the elements in an array you can divide
  - Size of the array by the size of the array type or an element; i.e. *sizeof(array) / sizeof(array[0])*

Arrays and Strings

- Number of elements in an array can be fixed or variable
  - Number of the elements in an array is also called **length** of the array
  - Fixed means the number of elements is known for the compiler during compilation
  - Variable means the number of elements is varying and specified during run time
    - We shall avoid using ***variable length arrays***
- By an **index** and the subscript (**[ ]**) operator we can get access to the elements
  - The index is an integer greater or equal to zero `int array[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}`
  - Indices start with **zero** and end with **length of the array - 1**
    - Index of the **first** element in an array is **zero**. E.g. array[0]
    - Index of the **last** element in an array is **length - 1**. E.g. array[9]
  - Name of an array **addresses** the first element in the array; i.e. **&array[0]. it is like a const pointer**

Arrays and Strings

- C does not provide boundary checking and you need to ensure that an index does not get out of the boundaries. E.g. values[9] = 0; is not OK
- By a loop you can go through a an array

```
int values[5] = { 1, 2, 3, 4, 5 };
for(int i = 0; i < (sizeof(values)/sizeof(int)); ++i)
{
     printf("Value=%d\n", values[i]);
}
```

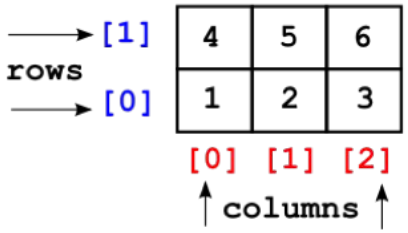
- You know the general rule of initialization
  - If a variable is global or the storage class of the variable is static, then
    - it shall be initialized with a **constant** or the compiler initializes it to **zero**
  - If a variable is local, its value is unknown and you need to initialize it
- An array can be initialized using an initialization list or element designators
  - Using list: int array[5] = {1, 2, 3, 4, 5}; // array[0] is 1, array[1] is 2, …, and array[4] is 5
  - Using designator: int array[5] = {1, **[1]** = 2, 3, 4, 5}; // array[0] is 1, array[1] is 2, …, and array[4] is 5

Arrays and Strings

- If an array is static or global, its initializers must be constant expressions
  - Otherwise you can use variables in the initialization of the array
- It is necessary to specify number of the elements of an array when you initialize it in its declaration using an initialization list. E.g. `int array[ 5] = {1, 2, 3, 4, 5}; // The length is 5`
- If an array is partially initialized, the **uninitialized** elements are automatically set to **zero**
- Arrays shall not be partially initialized. Exceptions:
  - An initializer of the form **{ 0 }** may be used
  - Initialization consists **only** of designated initializers may be used
  - An array initialized using a **string literal** does not need an initializer for every element.
- A variable length array can not be initialized in its declaration

Arrays and Strings

- Arrays can be multidimensional, just add another subscript operator
  - For example: int matrix[2][3]; // A two dimensional array which has 2 rows and 3 columns
  - Means that elements of an array are themselves arrays
    - The elements of an **n** dimensional array are **(n-1)** dimensional arrays and sure when **n** is one, the elements are not arrays



- Using loops you can go through a multidimensional array
- You can initialize a multidimensional array according the rules described previously

```
for (int row = 0; row < 2; ++row)
{
    for (int column = 0; column < 3; ++column)
    {
        printf("matrix[%d][%d] = %d\n", row, column, matrix[row][column]);
    }
}
```

  - int matrix[2][3] = {0}; **or** int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
  - int matrix[2][3] = {{1, 2, 3}, {[0] = 4, [1] = 5}}; **or** int matrix[2][3] = {{1, 2, 3}, [1][0] = 4, [1][1] = 5, [1][2] = 6};

Arrays and Strings

- A **string** is a null(**'\0'**) terminated sequence of characters.
- In C there is no string type and a string is stored in an **array** of type **char**
  - E.g. char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; // Length of the string is 5
  - **Length** of a string is the number of characters before the null(**'\0'**) character
  - Always the array length is at least one element longer than the string length
- It is possible to initialize a string using a string literal. E.g. char str[] = "Hello";
  - In this case it is not necessary to terminate the string with null (**'\0'**)
- There are some useful functions used for string handling in **stdio.h** and **string.h**
  - E.g. sprintf, sscanf, gets, puts, and etc. in stdio.h
  - E.g. strlen, strcat, strcpy, strcmp, and etc. in string.h
  - ***Not all of them are safe!!***