Arrays


Canberra Institute
of Technology


aie
SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

Contents

- What are arrays
- Declaring arrays
- Using arrays
- Array sizes
- Initialising arrays
- Iterating through arrays
- Out of bounds
- Copying arrays
- Comparing arrays

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

What are arrays?

- An array is a series of elements of the same type.
  - An array cannot store objects of different types
  - E.g. No mixing floats with ints with chars.

- They are stored in a contiguous block of memory

Other Memory	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	Other Memory
0x00	0x04	0x08	...	...	...	...	...	0x...

A few rules

- Array naming conventions should follow the same rules as other variables.
- Arrays are also scoped just like other variables.
- The array size defines how many elements can be stored in the array (some limits exist).

Declaring Arrays

```
//Declare an array holding 100 integers
int numbers[100];
```

- We can specify array sizes directly like the example above.
- Note that you cannot use a variable to specify the array size.

Array Sizes

- The size in bytes of an array is dependant upon the type of object that the array stores.

- You must specify the number of items an array can hold
  - float fastestLapTimes[10];

- The actual size in memory of an array is the size of that object (in bytes) multiplied by the number of objects in that array.
  - In the above example, the size of the array is 10 x sizeof(float) – or 40 bytes.

Using arrays

- Arrays are **indexed from 0**
  - Therefore if an array has 10 elements, these are accessed via indices 0-9.

- To access a value in an array, the index is passed to the subscript operator [].

```
int numbers[100];

numbers[3] = 5;             //Set this value to 5
numbers[3] *= 2;            //Multiply it by 2
std::cout << numbers[3];    //Retrive and output to the console
```

The subscript operator

- The subscript operator [] can receive any integer value.
- It cannot take decimal / real values (an index of 1.3323 is pointless)
- The index can be given in the form of a calculation or as a variable.

```
numbers[3 * 2] = 10;
int index = 2;
numbers[index] = 4;
int aNumber = numbers[index];
int anotherNumber = numbers[index + 1];
```

Alternative Declarations

- Arrays, like other variables, can be initialised upon declaration.

```
//Declare an array holding 10 integers
int numbers[10] = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512};
```

- The difference with other variables is that the content must be defined within braces {} and values must be separated by a comma.

- Initialisation values are stored in left to right order.

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Alternative Declarations

- What do you think will happen in the following declaration?

```
int numbers[10] = { 1, 2, 3 };
```

- And the following?

```
int numbers[] = { 1, 2, 3 };
```

Iterating through arrays

- We often want to access the elements of an array sequentially.

- The most common way of doing this is by using a for loop:

```
int someArray[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for(int i = 0; i < 10; ++i)
{
    std::cout << someArray[i] << std::endl;
}
```

For each

- For each loops are specifically designed to work with collections of objects – such as arrays.

```
int someArray[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for(int i : someArray)
{
    std::cout << i << std::endl;
}
```

- We can think of this as: for each integer in someArray then do the statements with the braces.

- We give a label to the current integer that is being processed. In this case, i.

Out of bounds

- If an index is beyond the scope of the array, e.g. Trying to access element 100 in an array of 90 objects, then bad things can, and will happen!

```
//Declare an array holding 10 integers
int highScores[10];

//Later in code....

//If an improperly addressed element is being assigned to a variable
//The value returned will be undefined and in this scenario certainly
//not a valid score.
int highScore = highScores[15]; //Out of bounds!!
```

Boundaries

- Things will only get worse if an element that is beyond the scope of an array is attempted to be written to.

- This is called buffer overflow and often results in seemingly random behaviour...or a crash.

- Sometimes the debugger will detect a buffer overflow (but not always) resulting in one of the following errors:
  - *The stack/heap around 'array' was corrupted*
  - *Subscript index for 'array' is out of bounds*

Array Types

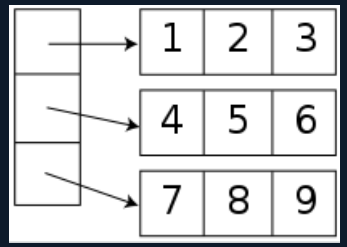
- Arrays can be of any type, so long as that type is defined somewhere within the scope of the project.

- These arrays are defined and accessed in the same way as integer arrays.

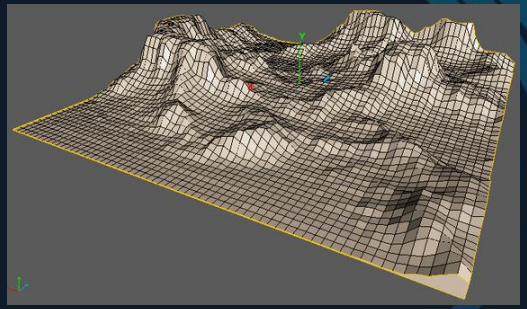
Arrays of arrays

- Since arrays can be of any type then it makes sense that we can have arrays of other arrays.

- These are called 2 dimensional arrays



2D Arrays



- 2 Dimensional arrays consist of both rows and columns of elements.
- For example we might have a simple 2D map with the value stored in the array describing the height of each point (a height map).
- The array consists of 3 rows and 3 columns
- To create an array to represent this format we declare it like this:
  - int map[3][3];

5	5	7
4	5	6
3	4	5

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

2D Arrays Continued

- 2D Arrays can be initialised on declaration just like 1D arrays. Just needs an extra set of '{ }‘

```
int map[3][4] =
{
    { 128, 256, 128, 64},
    { 64, 192, 256, 64 },
    { 0, 256, 128, 128 }
};
```

- You can think of it, as an array of arrays. In this case, 3 sets of integer arrays 4 elements long.

- The first number in the [] brackets represents the rows and the second represents the columns

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Accessing 2D array elements

```
std::cout << map[2][1];
```

- The code above will output the element on the third row and second column of the map array
- To iterate through a 2D array we usually use nested for loops

```
for (int row = 0; row < 3; ++row)
{
     for (int col = 0; col < 3; ++col)
     {
          std::cout << map[row][col];
     }
}
```

Storage

A multi-dimensional array is stored in memory as a *linear sequence of elements*, just like a one-dimensional array

	Other Memory				map[0][0]				map[0][1]				map[0][2]			
Bytes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	map[1][0]				map[1][1]				map[1][2]				map[2][0]			
Bytes	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	map[2][1]				map[2][2]				Other Memory				Other Memory			
Bytes	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

Multiple indices are mapped to a single index into memory

1D vs 2D

- It’s actually more common to use a 1D array that simulates the behaviour of a 2D array.

```
int map[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
int map2[9] = {  1, 2, 3, 4, 5, 6, 7, 8, 9 };
int row = 1, col = 2;
std::cout << map[row][col];
std::cout << map2[row * 3 + col];
```

- However while you are learning it may be easier to conceptualise a 2D array.

n-dimensional arrays

- You can have any number of dimensions in your array.
- You can have 3D arrays:
  - int voxelCube[3][4][6];
- 4D arrays:
  - int MyHeadHurts[3][4][5][6];
- And so forth, although they do get a little hard to visualise, but not impossible!

Assigning an array

- C++ **does not** permit a simple array to array copy using the assignment operator =.

- Elements must be copied individually, or copied as part of a block memory copy.

```
int highScores[10];
int newScores[10];

//Copy each element individually
for(int i = 0; i < 10; i++)
{
    newScores[i] = highScores[i];
}
```

Compare Arrays – Equal To

- Similarly, you cannot compare two arrays with the == operator
- Elements must be compared individually

```
int highScores[10];
int newScores[10];
/*
add some values into the arrays
*/
bool match = true;
for (int i = 0; i < 10; i++)
{
     if (newScores[i] != highScores[i])
     match = false;
}
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Outputting arrays

- When outputting an array we cannot do the following:

```
int highscores[10];

std::cout << highscores;
```

- Like comparing or copying, we need to output element by element:

```
int highscores[10];

for (int i = 0; i < 10; i++)
{
     std::cout << highscores[i] << " ";
}
```

Summary

- Arrays are extremely useful when we want to create a list or collection of objects of the same type.

- By using arrays and loops together, you can avoid needing to write large portions of code.

- Arrays are a bit trickier to pass into functions and you need to copy each individual element one at a time should you wish to copy an entire array.

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

References

- Gerdlsenberg, 2014*, Array*, Chess Programming Wiki, https://chessprogramming.wikispaces.com/Array

- Akenine-Moller, T, Haines, E & Hoffman, N 2008, *Real-Time Rendering*, 3rd edn, CRC Press

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX