Arrays I

DM2111
C++ Programming

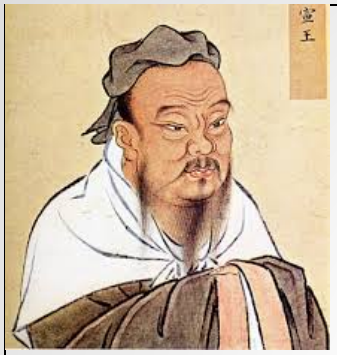
Introduction

Introduction	Break
Problem solving	Array and Strings
Basic elements of C++	Array and Strings
Basic elements of C++	Pointers
Statements	Pointers
Repetition	I/O operations
Functions	Structs
Functions	Others

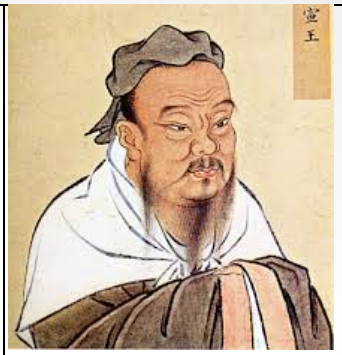
Agenda

- What is an array?
- Array Processing
- Array Parameters

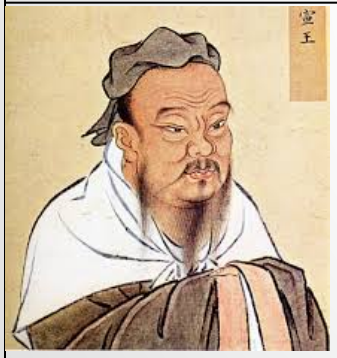
Problem solving



If you need one variable, declare one variable.



If you need two variables, declare two variables.



If you need three variables, declare three variables.

Problem solving

Say you want to add up 10 numbers

allocate memory for 10 variables

```
int a, b, c, d, e, f, g, h, i, j;
// some way to assign values to them
int sum = a+b+c+d+e+f+g+h+i+j;
```

What if there are more variables?

Ran out of variable names?

Enter the array

How can arrays help to solve the problem?

```
const size_t arrSize = 10;
int num[arrSize];
int sum = 0;
for (int i=0; i < arrSize; ++i)
{
    sum += num[i];
}
```

What is an array?

In memory, array takes up a continuous section.

```
int a, b, c, d, e, f, g, h, i, j;
```

	a	b	c		d	e	f		g
h		i	j						

```
int num[10];
```

NUM	1	2	3	4	5	6	7	8	9

What is an array?

An array is a fixed collection of data, all of the same data type

Declaring arrays

```
dataType name[size]
```

dataType - data type of the elements of the array

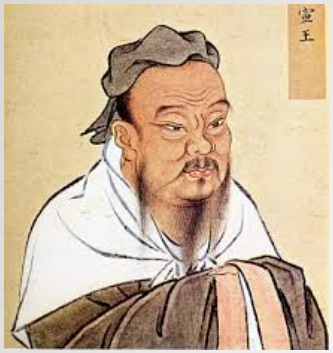
name - name of the array

size - size of the array. i.e. number of elements in the array. Must be a positive integer, known at compile time.

How to declare arrays?

```
int data[5];                    ✓

const int i = 10;
char name[i];                   ✓

const int k = 30;
double input[k * 2 + 25];       ✓

int arraySize;

cin >> arraySize;
int list[arraySize];            ✗
```

Accessing arrays



"Programmers start counting from 0"

```
const int i = 5;
char data[i];

data[2] = 'n';
data[3] = data[2];

data[i - 1] = 'y';

data[1] = data[2] - 9;
data[data[2] - data[2]] = data[1] + 6;
```

data[0]	k
data[1]	e
data[2]	n
data[3]	n
data[4]	y

Array Initialization

```
int array1[5] = {1, 2, 3, 4, 5};
int array2[]  = {1, 2, 3, 4, 5};
int array3[5] = {0};
int array4[5] = {56, 76, -4};
```

array1[0]	1	array2[0]	1	array3[0]	0	array4[0]	56
array1[1]	2	array2[1]	2	array3[1]	0	array4[1]	76
array1[2]	3	array2[2]	3	array3[2]	0	array4[2]	-4
array1[3]	4	array2[3]	4	array3[3]	0	array4[3]	0
array1[4]	5	array2[4]	5	array3[4]	0	array4[4]	0

Initialized with default values

Array use case

```
const int size = 5;
int values[size];
int sum = 0;
float avg = 0.0f;
int hIndex = 0;
int lIndex = 0;

for (int i = 0; i < size; ++i)
{
    cin >> values[i];
}

for (int i = 0; i < size; ++i)
{
    sum += values[i];
    if (values[i] > values[hIndex])
    {
        hIndex = i;
    }
    if (values[i] < values[lIndex])
    {
        lIndex = i;
    }
}
```

```
avg = static_cast<float>(sum) / static_cast<float>
(size);

cout << sum << endl;
cout << avg << endl;
cout << values[hIndex] << endl;
cout << values[lIndex] << endl;
```

values[0]	1
values[1]	2
values[2]	3
values[3]	4
values[4]	5

Array bounds

When accessing an array element, *array*[*index*], *index* is in bounds if it is between 0 and *arraySize* - 1; otherwise it is out of bounds

- tl:dr : 0 ≤ *index* < *arraySize*
- C++ does not check if *index* is in bounds
- If *index* is out of bounds, it may be accessing restricted memory locations
- It is **your**  responsibility to ensure *index* is within bounds

```
int values[5];

values[0];
values[-1];
values[4];
values[5];
values[6];
```

Arrays are processed per element in C++

Restriction – no aggregate operation (any operation that manipulates the entire array as a single unit)

```
int array1[3] = {1, 2, 3};
int array2[3];

array2 = array1; // does something entirely different
cin >> array2;
cout << array1;

for (int i = 0; i < 3; i ++)
{
    array2[i] = array1[i];
}
for (int i = 0; i < 3; i ++)
{
    cin >> array2[i];
}
for (int i = 0; i < 3; i ++)
{
    cout << array1[i];
}
```

Arrays are passed as pointers into functions

```
int sum (int num[], size_t size)
{
    int total = 0;

    for (size_t i = 0; i < size; ++i)
    {
        total += num[i];
    }
    return total;
}

int main (void)
{
    int values[] = {2, 4, 6, 8, 10};

    cout << sum(values, 5) << endl; // 30
}
```

Functions cannot return a value of type array

```
int[] triple (int a[]);
```

Possible to return a pointer to an array, but you need some arcane magic. Plus 20 mana to cast.

There are other easier ways to do it.

- Require calling function to pass in reference of array
- Use STL containers
- Objects