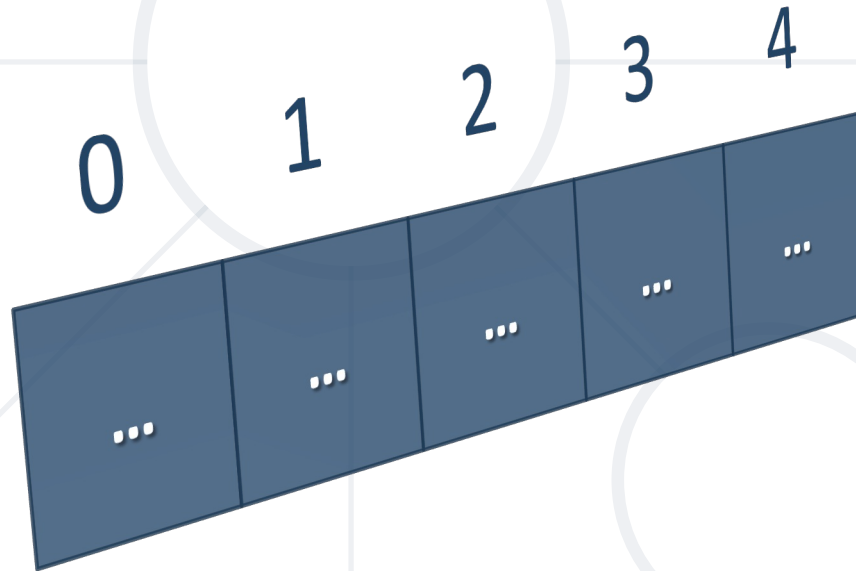


Arrays

Fixed-Size Sequences of Elements



SoftUni Team
Technical Trainers



SoftUni

Software University

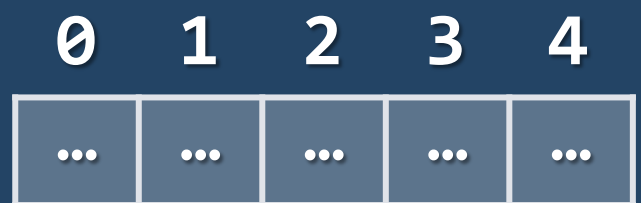
<https://softuni.bg/>

1. What are **Arrays**?
 - Creating Arrays
 - Declaring and Initializing
2. **Reading** and **Printing** Arrays
3. Usage with **Functions**
4. **Range-based** for loop
5. `<array>` Header



sli.do

#cpp-fundamentals



What are Arrays?

What Are Arrays?

- In programming, an **array** is a **sequence of elements**



- Elements are numbered from **0** to **Length-1**
- All elements are of the **same type**
- Arrays have **fixed size** and cannot be resized





Creating Arrays

- Declaring

```
{data type} {identifier}[{array size}];
```

- Arrays have some special **initialization syntax**

```
{data type} {identifier}[N] = {elem0, elem1, ..., elemN-1};
```

- There can be less than N elements, **but not more**

```
int numbers[5] = { 10, 9, 12, 31, 15 };
```

Index	0	1	2	3	4
Value	10	9	12	31	15



Declaring and Initializing

Array Declaration

- Array size must be an **integer**. Size can be:

- a **literal**
- a **constexpr**

```
double numbers[7];  
constexpr int NUM_LETTERS = 26;  
char alphabet[NUM_LETTERS];
```

- You can also declare an array **without a specified size**
 - The compiler is smart enough to get the elements' count we put inside the braces

```
int numbersToFive[] = { 1, 2, 3, 4, 5 };
```



- **{ }** initializes elements (comma-separated values)
 - if less values than array size: **remaining get default values**
 - if more values than array size: **compilation error**

```
double values[3] = {3.14};  
double sameValues[3] = {3.14, 0, 0};
```

- Other rules are the same as for primitives:
 - Can only be **initialized once**
 - Can be made **const**



Reading and Printing Arrays

Accessing Array Elements

- The indexing operator `[]` gives access to any array element

```
array[{index}] = 10;  
int value = array[{index}];
```

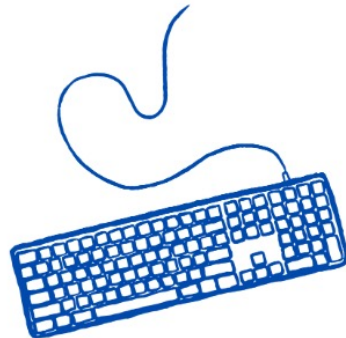
- Once you access the element, treat it as a **normal variable**

```
int array[3] = { 20, 30, 40 };  
cout << "First->" << array[0]  
      << "Second->" << array[1]  
      << "Third->" << array[2] << endl;
```



- Arrays are often read-in from some input, **instead of initialized**
- Common approach: **run a loop to read-in a number of elements**
 - Example: read-in a specified number of elements from console

```
for (int i = 0; i < n; i++)  
{  
    cin >> arr[i];  
}
```



Printing an Array

- You will commonly need to display **all elements of an array**
- Common approach: **loop over the elements, print each**

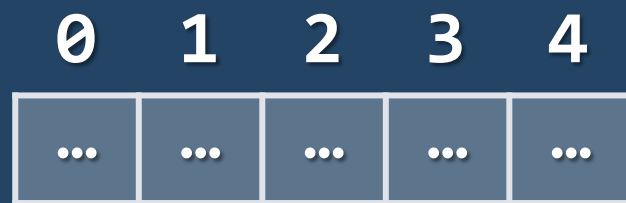
```
for (int i = 0; i < n; i++)  
{  
    cout << arr[i] << endl;  
}
```



Read and Print an Array

```
const int length = 20;


int array[length];
cout << "Enter elements in array: " << endl;
for (int i = 0; i < length; i++)
{
    cin >> array[i];
}
cout << "Elements in array: " << endl;
for (int i = 0; i < length; i++)
{
    cout << array[i] << " ";
}
```



Arrays as Function Parameters

Arrays as Function Parameters

- Array parameters are declared the same way arrays are declared
 - Usually necessary to add an **int** parameter with the size



```
void print(int array[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << array[i] << " ";
    }
    cout << endl;
}
```

```
int main()
{
    int numbers[] = {1, 2, 3};
    print(numbers, 3);
    return 0;
}
```

- Functions work with the original array the caller uses
 - If the function **changes an element**, the caller's array is modified
 - Array elements are passed **by reference**
- Functions can't return **static** arrays created in them
 - Arrays are essentially **memory addresses**
 - The memory they point to **is freed when the function exits**



Arrays as Pointers

Arrays as Pointers

- **Pointer** - a variable that holds the **address** of another variable in the memory
 - The address stores its **value**
 - Pointers have **data type**
- **Arrays** can be represented as pointers
 - The array is a **sequence of variables stored in memory**
 - The array name points to the first item



Arrays as Pointers

- Examples:

```
int *ptr;  
int arr[4];  
ptr = arr;
```

```
ptr + 0 is equivalent to &arr[0];  
ptr + 1 is equivalent to &arr[1];  
ptr + 2 is equivalent to &arr[2];  
ptr + 3 is equivalent to &arr[3];
```

```
*ptr == arr[0];  
*(ptr + 1) is equivalent to arr[1];  
*(ptr + 2) is equivalent to arr[2];
```





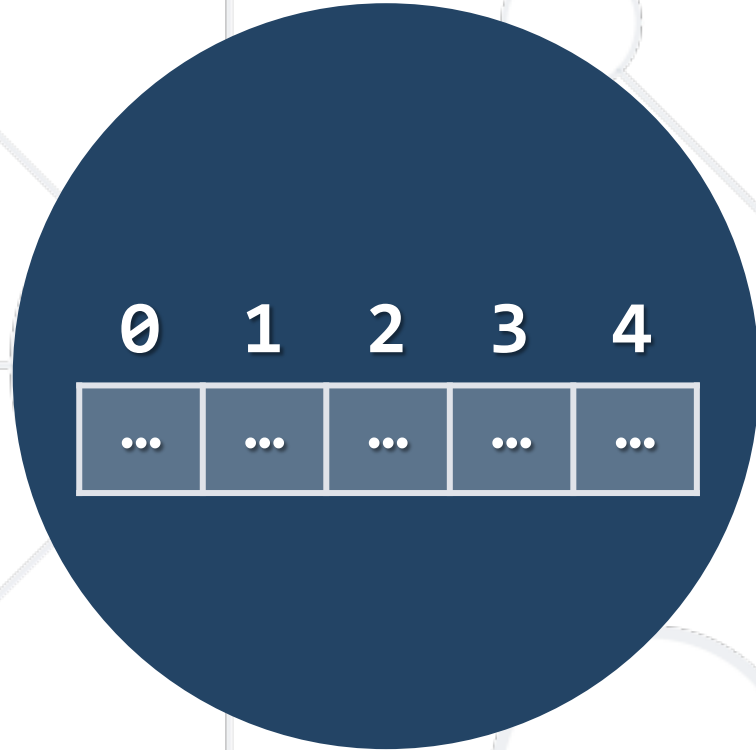
Range-Based For Loop

- Syntax:

for(const DataType& element : array)

- Body will execute **once for each element** in the array
- On each iteration, **element** will be the next item in the array

```
int numbers[] = { 13, 42, 69 };  
for (int num : numbers)  
{  
    cout << num << endl;  
}
```



<array> Header

- The **array** class knows its size, can be returned from functions

#include<array>

- Declaring and Initializing:

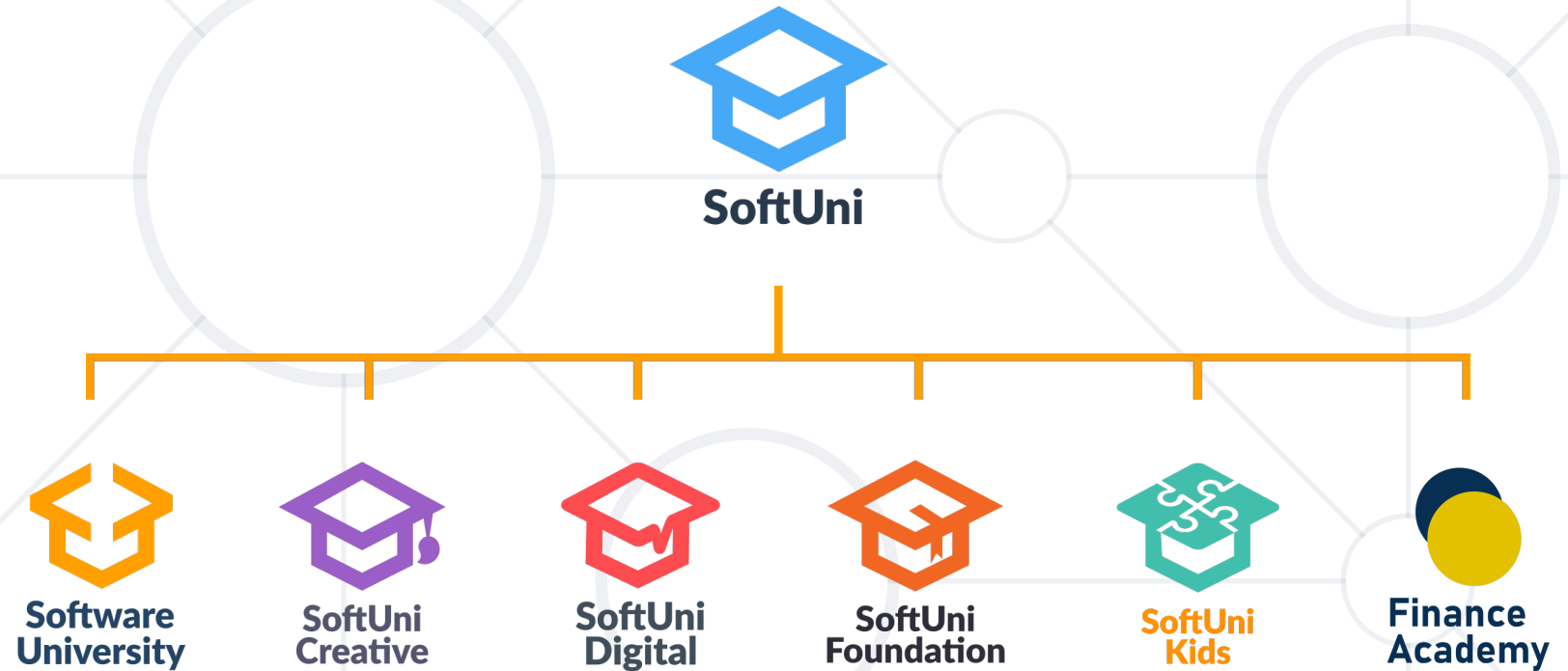
```
array<int, 5> arr = { 1, 2, 3, 4, 5 };
```

- **arr.size()** gives you the size of the array
- Use the **[]** operator like with normal arrays to access elements

- Arrays hold a **sequence** of elements
 - Elements are numbered from **0** to **length-1**
- **Creating** an array
- Accessing array elements by **index**
- **Printing** array elements
- Range-based **for loop**



Questions?



SoftUni Diamond Partners



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg, about.softuni.bg
- Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity

