Array Initialization and Basic Array Operations

This lesson explains how arrays can be initialized and the basic operations that can be performed on elements of an array.

WE'LL COVER THE FOLLOWING

Initializing the elements

Like every variable in D, the elements of arrays are automatically initialized. The initial value of the elements depends on the type of the elements: 0 for `int`, double.nan for `double`, etc.
All of the elements of the values array above are initialized to double.nan:

```
double[5] values; // elements are all double.nan
```

Obviously, the values of the elements can be changed later during the execution of the program. We have already seen that in the previous lesson when assigning to an element of an array:

```
monthDays[11] = 31;
```

That also happened when reading a value from the input:

```
readf(" %s", &values[counter]);
```

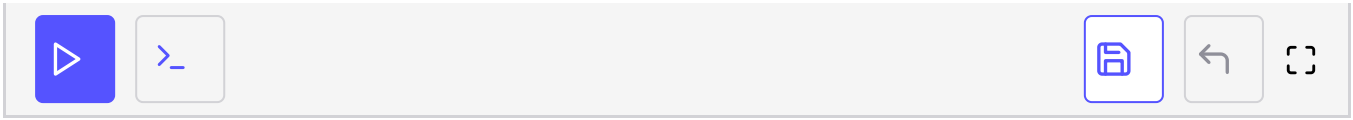
Sometimes the desired values of the elements are known at the time the array is defined. In such cases, the initial values of the elements can be specified on the right-hand side of the assignment operator within square brackets. Let's see this in a program that reads the number of the month from the user and prints the number of days in that month:

```
import std.stdio;

void main() {
    // Assuming that February has 28 days
    int[12] monthDays =
        [ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 ];

    write("Please enter the number of the month: ");
    int monthNumber;
    readf(" %s", &monthNumber);
    if(monthNumber < 1 || monthNumber > 12) {
        writeln("wrong input");
    }

    else {
    int index = monthNumber - 1;
    writeln("Month ", monthNumber, " has ",
            monthDays[index], " days.");
    }
}
```



Days in a month

As you can see, the `monthDays` array is defined and initialized at the same time. Also note that the number of the month, which is in the range 1-12, is converted to a valid array index in the range 0-11. Any value that is entered outside of the 1-12 range would cause the program to be terminated with an error.

When initializing arrays, it is possible to use a single value on the right-hand side. In that case all of the elements of the array are initialized to that value:

```
int[10] allOnes = 1; // All of the elements are set to 1
```

Basic array operations

Arrays provide convenience operations that apply to all of their elements.

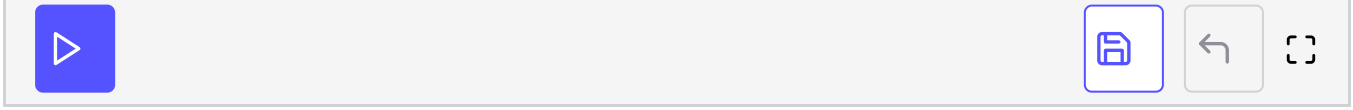
Copying fixed-length arrays

The assignment operator copies all of the elements from the right-hand side to the left-hand side:

```
import std.stdio;

void main() {
    int[5] source = [ 10, 20, 30, 40, 50 ];
    int[5] destination;
    destination = source;

    writeln("Source array: ",source);
    writeln("Destination array: ",destination);
}
```



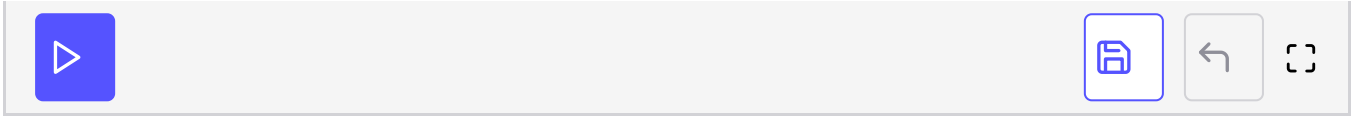
Adding elements to dynamic arrays

The `~=` operator adds new elements to the end of a dynamic array:

```
import std.stdio;

void main() {
    int[] array;          // empty
    array ~= 7;           // array is now equal to [7]
    array ~= 360;         // array is now equal to [7, 360]
    array ~= [ 30, 40 ]; // array is now equal to [7, 360, 30, 40]

    writeln(array);
}
```



It is not possible to add elements to fixed-length arrays:

```
int[10] array;
array ~= 7; // ← compilation ERROR
```

Removing elements from dynamic arrays

Array elements can be removed with the `remove()` function from the `std.algorithm` module. `remove()` cannot actually change the number of elements of the array. Rather, it has to move some of the elements of the array

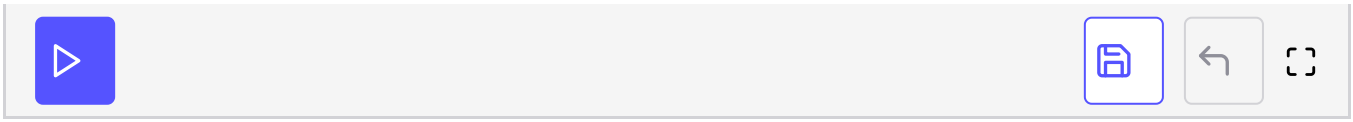
one or more positions to the left. For that reason, the result of the remove

operation must be assigned back to the same array variable.
There are two different ways of using `remove()`:

1. Providing the index of the element to remove. For example, the following code removes the element at index 1.

```
import std.stdio;
import std.algorithm;

int main() {
    int[] array = [ 10, 42, 20, 30, 42, 40 ];
    array = array.remove(1);                // Assigned back to array
    writeln(array);
    return 0;
}
```

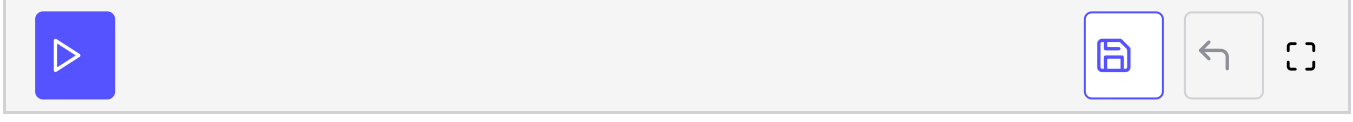


Code to remove element from an array

2. Specifying the elements to remove with a lambda function, which we will cover in a later chapter. For example, the following code removes the elements of the array that are equal to 42.

```
import std.stdio;
import std.algorithm;

void main() {
    int[] array = [ 10, 42, 20, 30, 42, 40 ];
    array = array.remove!(a => a == 42);    // Assigned back to array
    writeln(array);
}
```



Code to remove element from an array

Combining arrays

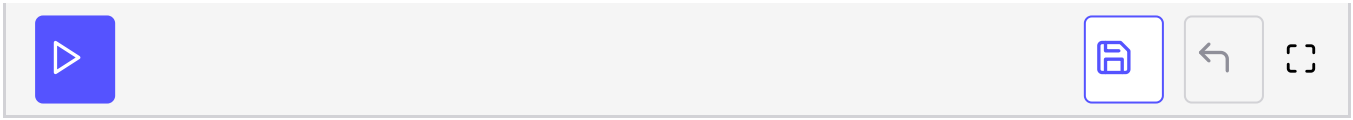
The `~` operator creates a new array by combining two arrays. Its `~=` counterpart combines the two arrays and assigns the result back to the left-hand side array:

```
import std.stdio;

void main() {
    int[10] first = 1;
    int[10] second = 2;
    int[] result;

    result = first ~ second;
    writeln(result.length);     // prints 20

    result ~= first;
    writeln(result.length);     // prints 30
}
```



Combining arrays

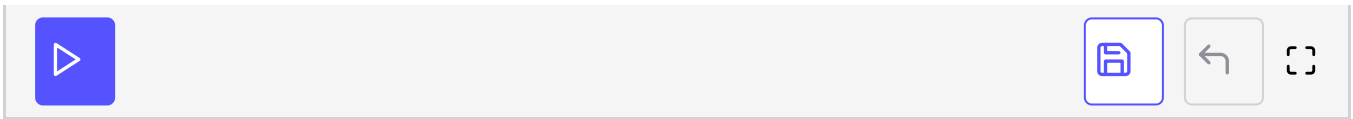
The `~=` operator cannot be used when the left-hand side array is a fixed-length array:

```
int[20] result;
// ...
result ~= first; // ← compilation ERROR
```

If the array sizes are not equal, the program is terminated with an error during assignment:

```
import std.stdio;

void main() {
    int[10] first = 1;
    int[10] second = 2;
    int[21] result;
    result = first ~ second;
}
```

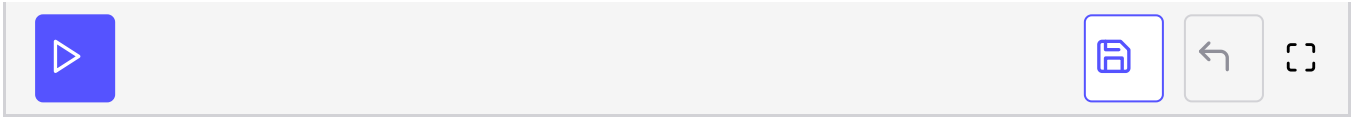


Sorting the elements

`std.algorithm.sort` can sort the elements of many types of collections. In the case of integers, the elements get sorted from the smallest value to the greatest value. In order to use the `sort()` function, one must import the `std.algorithm` module first.

```
import std.stdio;
import std.algorithm;

void main() {
    int[] array = [ 4, 3, 1, 5, 2 ];
    sort(array);
    writeln(array);
}
```



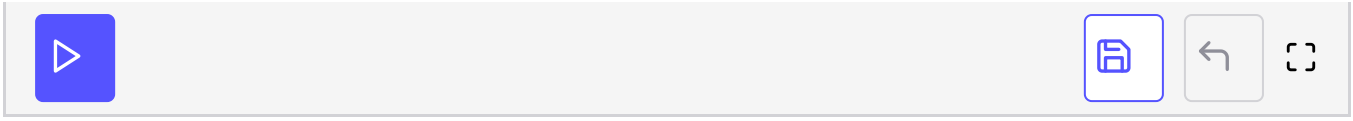
Sorting the elements

Reversing the elements

`std.algorithm.reverse` reverses the elements in place (the first element becomes the last element, etc.):

```
import std.stdio;
import std.algorithm;

void main() {
    int[] array = [ 4, 3, 1, 5, 2 ];
    reverse(array);
    writeln(array);
}
```



Reverse the elements

In the next lesson, we will look into associative arrays, how they are different from plain arrays and how we can use some important properties of associative arrays.