



Dynamic Arrays

Learn how to allocate memory to an array dynamically.

We'll cover the following

^

- Dynamic arrays
 - Declaration
 - Initialization
 - Deallocating array
 - Print all elements of a dynamic array
 - Resizing dynamic array
 - Explanation

Dynamic arrays#

In the arrays lesson, we discussed static arrays. In a static array, a fixed amount of memory is allocated to the array during the compile time.

Therefore, we cannot allocate more memory to the arrays during program execution.

Suppose we have declared an array that can store five integer values.

What if we want to store more than five values in an array? Here is dynamic arrays come in!

Dynamic arrays can grow or shrink during the program execution.





Declaration#

The general syntax for declaring dynamic arrays is given below:

```
DataType *ArrayName = new DataType [size] ;
```

Initialization#

We can initialize a dynamic array just like a static array.

```
ArrayName [ Index ] = Value ;
```

Deallocating array

The basic syntax for deallocating a dynamic array is given below:

```
delete [] ArrayName;
```

Print all elements of a dynamic array#

The code will initialize the dynamic array and then print all its elements using the for loop.

```
1 #include <iostream>
2
3 using namespace std;
```

```
5 int main(){
                                                              €€}}
      int size = 5;
 7
      //Declare dynamic array
      int *Array = new int[size];
      //Initialize dynamic array
      for(int i = 0; i < size; i++){
10
         Array[i] = i;
11
12
      }
13
      //Prints dynamic array
14
       for(int i = 0; i< size; i++){
15
         cout << Array[i] << " ";
16
      }
17
     // Deletes a memory allocated to dynamic array
        delete[] Array;
18
19 }
                                                             \triangleright
                                                                           X
                                                                       0.95s
Output
 0 1 2 3 4
```

Line No. 8: Reserves space for 5 integers and stores the starting address of allocated space in a pointer Array.

Line No. 10: Traverses an array and initializes its elements.

Line No. 14: Traverses an array and prints its value.

Line No. 18: Frees the space pointed by an Array.

Resizing dynamic array

Let's write a program in which we will increase the size of an array and store





```
// main function
14 int main() {
15
    // Initialize variable size
16
     int size = 5;
17
    // Create Array
    int * Arr = new int[size];
18
19
     // Fill elements of an array
     for (int i = 0; i < size; i++) {
20
21
      Arr[i] = i;
22
     }
23
     // Call printArray function
24
     printArray(Arr, size);
     // Create new array
25
     int * ResizeArray = new int[size + 2];
26
27
     // Copy elements in new arary
28
     for (int i = 0; i < size; i++) {
       ResizeArray[i] = Arr[i];
29
30
     }
31
     // Delete old array
32
     delete[] Arr;
33
     // Pointer Array will point to ResizeArray
34
     Arr = ResizeArray;
     // Store new values
36
     Arr[size] = 90;
37
     Arr[size + 1] = 100;
38
     // Call printArray function
     printArray(Arr, size + 2);
39
40 }
```

Explanation#

Suppose we want to resize the already declared array Arr in a program. We will follow the following steps:

Line No. 26: Creates a new array ResizeArray with the new size.

Line No. 28: Copies the elements of the old array Arr into the new array ResizeArray.

Line No. 32: We don't need the memory pointed by the Arr anymore. So, we

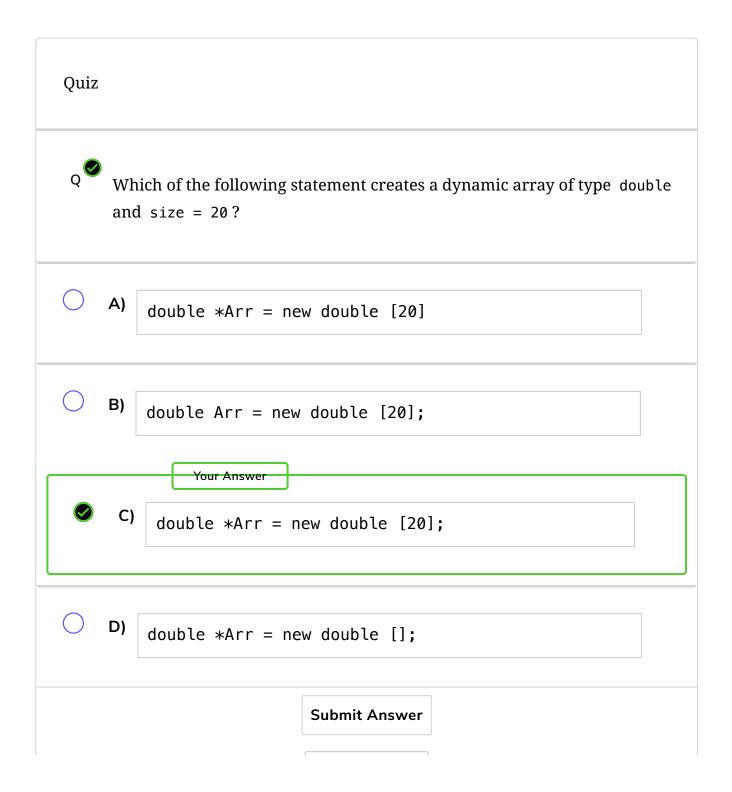


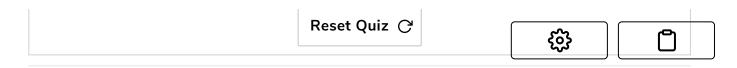


Line No. 34: We still want our array to be called Arr. Therefore we change the pointer.

Line No. 36: Stores the new values in an array Arr.

Tada! We have just resized the original array using pointers.





That's all about dynamic memory allocation. Let's do a few coding challenges in the upcoming lessons.

