Solution Review: Delete an Element at a Specific Index

Let's go over the solution review of the challenge given in the previous lesson.

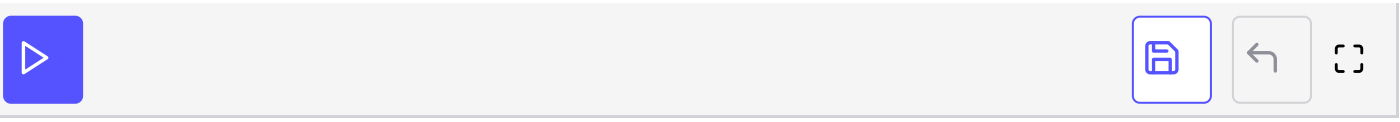
We'll cover the following

Solution

Press the **RUN** button and see the output!

```
#include <iostream>

using namespace std;

// printArray function
void printArray(int * arr, int size) {
  for (int i = 0; i < size; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
}

// delete_element function
void delete_element(int *&arr, int size, int index) {
  // Declare new array dynamically
  int * new_arr = new int[size - 1];
  // Traverse array
  for (int i = 0; i < size - 1; i++) {
    //
    if (i == index || i > index) {
      new_arr[i] = arr[i + 1];
    }
    else {
      // Copy elements in new array
      new_arr[i] = arr[i];

    }
  }
  // Free memory pointed out by arr
  delete[] arr;
  // Pointer arr will point to new_arr
  arr = new_arr;
  //return arr;
}
```

```
// main function
int main() {


  // Initialize variables
  int size = 5;
  int index = 3;
  // Initialize dynamic array
  int * arr = new int[size];
  // Fill elements in an array
  for (int i = 0; i < size; i++) {
    arr[i] = i;
  }
  // Call printArray function
  printArray(arr, size);
  // Call delete_element function
  delete_element(arr, size, index);
  // Call printArray function
  printArray(arr, size - 1);

  return 0;
}
```



Explanation

To delete the element at the given index, we copy the elements before the given index in a new array. However, when we reach the given index, we left shift the rest of the values in a new array. In this way, we delete the element at the given index.

delete_element function

The `delete_element` function takes a pointer to the `int` array in its input parameters. It also takes the values for `size` and `index`.

First, declare a new array dynamically of a size equal to `size-1` as we will be deleting one element. Traverse the original array. If the index `i` is less than the `index` to be deleted, simply copy the elements from the original array to a new array **(Line No. 25)**. When `i` becomes equal to `index`, ignore the `index` element and fill each element of a new array with the next element of the original array. Free the memory pointed out by `arr` and point `arr` to `new_arr`.

Let's wrap up this chapter by completing a quiz in the next lesson.