1 C-Strings

Kompilieren Sie folgende Applikation:

```
1  int main() {
2    char msg[10];
3    char *p;
4    char msg2[]= "Hello";
5
6    msg = "Bonjour";
7    p   = "Bonjour";
8
9    msg = p;
10   p = msg;
11 }
```

Sie werden bemerken, dass der Compiler zwei Fehlermeldungen erzeugt. Erklären Sie warum.

1.1 Lösung

```
1  int main() {
2    char msg[10]; // array of 10 chars
3    char *p;      // pointer to a char
4    char msg2[]= "Hello"; // msg2 = 'H''e''l''l''o''\0'
5
6    // ERROR: cannot convert from 'const char [8]' to 'char [10]'
7    msg = "Bonjour";
8
9    // address of “Bonjour” goes into p
10   p   = "Bonjour";
11
12   // ERROR: cannot convert from 'char *' to 'char [10]'
13   msg = p;
14
15   // OK
16   p = msg;
17 }
```

2 Arrays als Funktions-Parameter

Implementieren Sie die Funktion `FillArray`, welcher Sie ein 1-D Array und seine Länge als Parameter übergeben können. Die Funktion soll den Inhalt des Arrays verändern und einen Pointer zum ersten Element des veränderten Arrays zurückgeben. Folgend der Aufrufende Code und die Funktions-Signatur.

```cpp
void TestFillArray() {
  static constexpr size_t kArraySize = 10;
  int y[kArraySize] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int* a = FillArray(y, kArraySize);

  std::cout << "-- TestFillArray --" << std::endl;
  for (size_t i = 0; i < kArraySize; ++i) {
    std::cout << a[i] << ", ";
  }
  std::cout << std::endl;
  std::cout << "--------------------------" << std::endl;
}
```

```cpp
#include <cstddef>

int* FillArray(int* array, size_t array_size);
```

2.1 Lösung

```cpp
int* FillArray(int* array, size_t array_size) {
  for (size_t i = 0; i < array_size; ++i) {
    array[i] = 42;
  }
  return array;
}
```

3 C-Strings und Pointerarithmetik

Schreiben Sie eine Vergleichsfunktion für zwei C-Strings. Die Funktion soll zwei Pointer auf Character-Arrays entgegennehmen und **true** zurückgeben, falls die beiden C-Strings die gleiche Länge haben. Sie dürfen diese Aufgabe nicht mit `std::string` lösen.

Nehmen Sie die nachfolgenden C-Strings, um Ihr Programm zu testen.

```cpp
char string0[] ={'J', 'a', 'a', 'v', 's', 's', '\0' };
char string1[] ={'J', 'a', 'a', 'v', 's', '\0'};
char string2[] = "Jaavs";
char string3[] ={'J', 'a', 'a'};
```

Folgend die Signatur der Vergleichsfunktion.

```cpp
bool CompareStringLength(const char* ptr0, const char* ptr1);
```

3.1 Lösung

```cpp
#include "compare_string.h"

#include <iostream>

bool CompareStringLength(const char* ptr0, const char* ptr1) {
  // with pointer arithemtic
  size_t i = 0;
  while (*ptr0++ != '\0') {
    i++;
  }

  // with index operator
  size_t j = 0;
  while (ptr1[j]) j++;

  return i == j;
}

void TestCompareString() {
  char string0[] = "Ohhhhhhhhhhhhhhhhhhhhhh";
  char string1[] = {'J', 'a', 'a', 'v', 's', '\0'};
  char string2[] = "Jaavs";

  std::cout << "-- TestCompareString --" << std::endl;

  std::cout << "string1 and string2 are of the same length: "
            << std::boolalpha
            << CompareStringLength(string1, string2) << std::endl;
  std::cout << "string0 and string2 are of the same length: "
            << std::boolalpha
            << CompareStringLength(string0, string2) << std::endl;
  std::cout << "string0 and string1 are of the same length: "
            << std::boolalpha
            << CompareStringLength(string0, string1) << std::endl;

  std::cout << "--------------------------" << std::endl;
}
```