Introduction to Computing – Lab

Faculty of Information Technology & Computer Science

Lab 13
Course Instructor: Fraz Aslam
Topics: Character Arrays

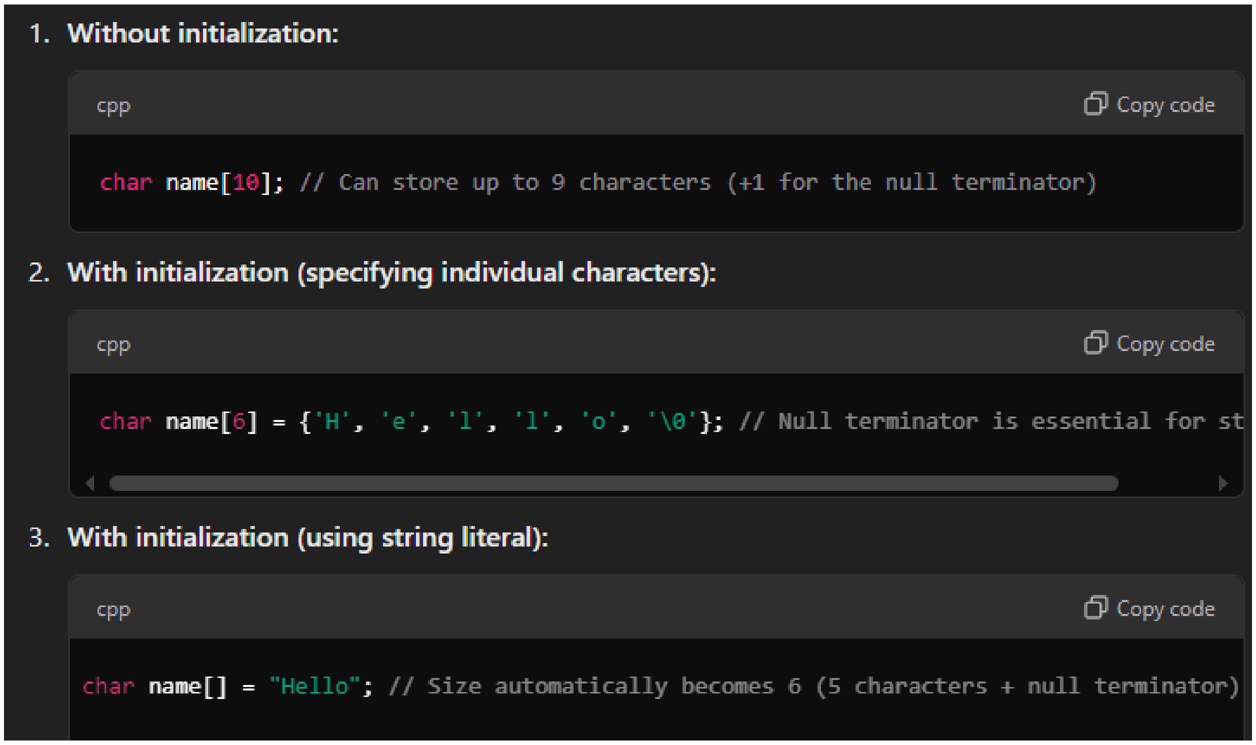
Instructions:

- Create separate C++ source files for each task, named as “task1.cpp”, “task2.cpp”, and so on depending on the task number.
- After completing all tasks, place all .cpp files into a folder. Name the folder with your university registration number (e.g., L1F20BSCS0999). Compress the folder into a .zip file and upload it on the portal.
- Ensure that the work you submit is entirely your own. Avoid copying from peers, online sources, or any other unauthorized material. Plagiarism will not be tolerated.
- If you encounter difficulties, feel free to reach out to the instructor. Collaboration and discussion are encouraged, but the final implementation should be your own work.
- Write clean and well-structured code. Use comments to explain key sections of your code to make it easier for others (and yourself) to understand.
- These tasks are designed to help you strengthen your logical thinking and problem-solving skills. Think through each problem carefully before starting to code. The aim is to develop a deep understanding of the problem and to devise solutions independently.
- Learning programming is about practice and perseverance—genuine effort in solving the problems will contribute significantly to your learning.

Character Arrays

A character array is a collection of characters stored in contiguous memory locations. Character arrays are commonly used for storing strings in C++. The size of a character array must be specified at the time of declaration and cannot be changed afterward.

Declaring and Initializing a Character Array



1. **Without initialization:**

```cpp
char name[10]; // Can store up to 9 characters (+1 for the null terminator)
```

2. **With initialization (specifying individual characters):**

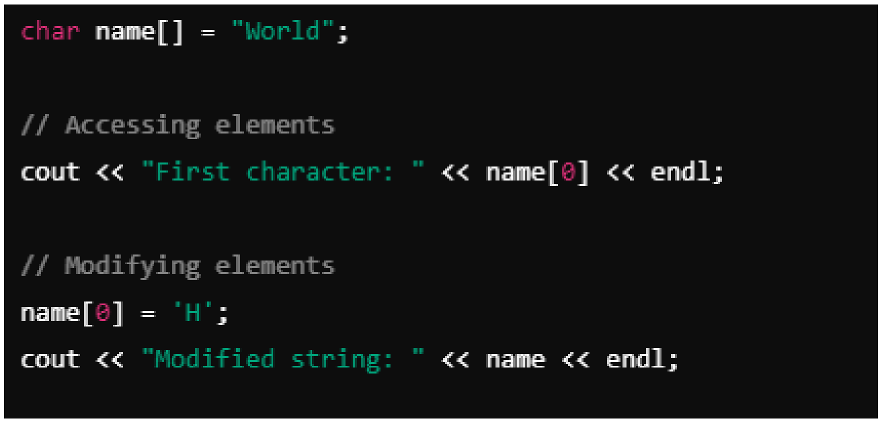
```cpp
char name[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; // Null terminator is essential for st
```

3. **With initialization (using string literal):**

```cpp
char name[] = "Hello"; // Size automatically becomes 6 (5 characters + null terminator)
```

Accessing and Modifying Individual Elements

Array elements are accessed using their index (starting from 0).



```cpp
char name[] = "World";

// Accessing elements
cout << "First character: " << name[0] << endl;

// Modifying elements
name[0] = 'H';
cout << "Modified string: " << name << endl;
```

Partial Initialization

Uninitialized elements are automatically set to \0 (null character).

```
char letters[5] = {'A', 'B'}; // Rest of the array is set to '\0'
```

```
letters[0] = 'A'
letters[1] = 'B'
letters[2] = ''
letters[3] = ''
letters[4] = ''
```

Out of Bound Index

Accessing an index outside the array's size results in undefined behavior. The program may crash or print garbage values.

```
int main() {
    char name[] = "Hello";

    cout << "Out-of-bound access: " << name[10] << endl; // Undefined behavior

    return 0;
}
```

Size of an Array

To get the size of an int array in C++, divide the total size of the array by the size of a char:

```
int totalSize = sizeof(name);        // Total size in bytes
int charSize = sizeof(char);         // Size of a single char
int arraySize = totalSize / charSize;
```

Taking Sentence as an Input

To store a sentence (with spaces) into a char array we have to use cin.getline():

```
char sentence[200];
int wordCount = 1;

cout << "Enter a sentence: ";
cin.getline(sentence, 200);
```

Problem with cin and cin.getline

When you use cin (e.g., cin >> variable) to input a value, it leaves a newline (\n) character in the input buffer after the user presses Enter.:

```
int num;
char sentence[100];

cin >> num;            // Input: 42 (Enter)
cin.getline(sentence, 100);
```

cin >> num reads the number 42 and stops reading when it encounters the newline (\n). The newline (\n) remains in the input buffer. When cin.getline is called, it sees the leftover \n in the buffer, assumes it's the input, and stops immediately without reading the intended string.

Solution: Using cin.ignore

The cin.ignore() function clears the unwanted characters in the input buffer. Typically, you use it after cin and before cin.getline to discard the leftover newline character.

```
int num;
char sentence[100];

cin >> num;              // Input: 42 (Enter)
cin.ignore();            // Discards the newline character
cin.getline(sentence, 100); // Now reads the intended input
```

Task 1

Write a program that removes all spaces from a given character array.

Task 2

Write a program to find common elements between two character-arrays.

Task 3

Write a program to find and print unique elements in a character array.

Task 4

Write a program that counts the frequency of each character in a character array.

Task 5

Write a program in which two character-arrays are taken from the user and merged into a third array. But the entries are done in alphabetical order (sorted).

Task 6

Write a program in which the user enters names of animals in a character array of size 100 (with spaces). Then the user enters an animal's name in second array and the program determines if the second array is present in the first array.

Task 7

Write a program that takes a sentence as input from the user. It then separates capital letters, small letters, numbers, and special characters in four separate character arrays.

Task 8

Write a program that finds a specific character and replaces it with another character.