Workshops 3, 4. Strings and Symbolic Data Types

G. Zhulikov

January 18, 2021

Rev. 1.1 as of 18.01.2021

1 Prerequisites, Goals, and Outcomes

Prerequisites Students should have mastered the following prerequisite skills: Building simple C++ programs. Using loops and `if`-statements in C++ programs.

Goals Learning how to work with types `char`, an array of `char` and `std::string` and how to include them in a C++ program.

Outcomes Students successfully completing this assignment would master the following: building a program that uses types `char`, an array of `char` and `std::string`.

2 Description

C++ has several data types representing characters and strings.

2.1 Char

The type `char` represents a single character. It is the basic type that is the foundation of the string types. While other string types may contain strings with several characters, it is important to make a distinction between them and the basic `char` type that contains only a single character.

You can use `char` objects the following way:

Listing 1. Using char type.

```
1    // Creating and initializing a variable of the type char
2    char c = 'A';
3
4    // Using a char literal ('y' in this case)
5    std::cin >> c;
6    if (c == 'y')
7    {
8        // do something
9    }
```

An important this to remember is how type casting works with characters. Every character is stored as a number representing it in a certain encoding table. For example, the code for '0' is 48 and the code for 'A' is 65. When you convert these values into integer types, the numbers stay the same and only the type of the varible changes.

2.2 do…while loop

In addition to the `for` loop and the `while` loop C++ has a `do...while` loop that is similar to `while` but the condition is checked after the loop iteration. It may be useful when you need to know if you should stop your program or do another iteration based on the some information received during the iteration.

Listing 2. A do...while loop.

```
char ans;
do {
    // problem
    cout << ”Press y to repeat”;
    cin >> ans;
} while(ans == ’y’ || ans == ’Y’);
```

2.3 C-style strings

C-style strings are arrays of `char` values. In practice they should be used only as string literals - constant string values.

Here is an example of a context where a C-style string would be appropriate.

Listing 3. C-style string literal example.

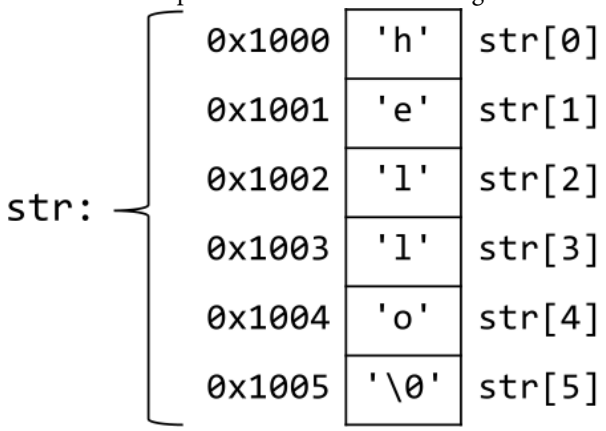
```
cout << ”Hello”;
```

The type of `”Hello”` in this example is an array of `char` values, so you can assign it to a variable that also has that type.

Listing 4. C-style string assignment.

```
char exampleString[] = ”Hello”;
```

This line of code means that an array `exampleString` is created and its length is inferred from the value assigned to it. The length of this array will be equal to the length of the string + 1. The reason for this is that C-style strings have a `’0’` character which represents the end of the string.



For more in-depth explanation of this topic you may refer to the *C++ Primer Plus* [1] book by Stephen Prata.

[1]

2.4 std::string

If you want to work with real strings in your program, do not use C-style strings. Use the high-level data type `std::string` instead.

The `std::string` data type provides useful and convenient methods for working with strings. For example, if a variable called `hello` has a type `std::string`, you can do the following:

1. Get the length of the string using `hello.length()`.
2. Concatenate two strings using addition operator.
   `largeString = hello + shortString;`.
3. Compare two strings using regular logical operators.
   `if (hello < shortString)`.
4. Use the last two operators with string literals.
   `largeString = hello + ”World”;`
   `if (hello < ”World”)`.
5. Copy strings using assignment.
   `hello = shortString;`.

Listing 5. std::string example.

```
#include <iostream>
#include <string>

int main()
{
    using namespace std;
    string exampleString = ”Hello”;
    string anotherString = ”World”;


    int exampleLength = exampleString.length();
    char firstLetter = exampleString[0];
    char lastLetter = exampleString[exampleLength - 1];

    string fullString = exampleString + anotherString;

    cout << ”The strings are:” << endl;
    cout << exampleString << endl;
    cout << anotherString << endl << endl;
    if (exampleString < anotherString)
    {
        cout << exampleString << ” < ” << anotherString << endl;
    }
    else if (exampleString > anotherString)
    {
        cout << exampleString << ” > ” << anotherString << endl;
    }
    else
    {
        cout << ”The strings ” << exampleString;
        cout << ” and ” << anotherString << ” are equal” << endl;
    }
    cout << ”The length of ” << exampleString << ” is ” << ▼
 exampleLength << endl;
    cout << ”The combined string is ” << fullString << endl;
    cout << ”The first letter in ” << exampleString;
    cout << ” is ” << firstLetter << endl;
```

```
37          cout << "The last letter in " << exampleString;
38          cout << " is " << lastLetter << endl;
39      }
```

3 Files

All files needed to master this task are available at an associated github-repository[2]. The structure of a project is described in embedded `.md` documents[3].

[2] https://github.com/pumpkin-code/dsba-prog-2021spring-wshps03-04

[3] See `README.md` in the root for the begining.

4 Tasks

1. Create a program that takes two numbers as input $1 < m, n < 8$. Output a multiplication table $m * n$, aligning output at tab positions.
2. Create a program that asks the name of the user and then outputs `"Hello, <user>"` where `<user>` is the name the user inputted into the program.
3. *For independent study.* Recreate the multiplication table task, allowing the user to rerun the program again after it prints the multiplication table. Use a new project for this task, do not erase the project containing the multiplication table program without reruns.
4. Count the length of a C-style string variable initialized with a string literal. Use index operator `[]` and a counter `i`.
   ```
   char current = exampleString[i];
   ```
   Stop counting when `current == '\0'`. At this point `(i - 1)` equals string length. Print it in the following format:
   `String <yourString>. Length is <stringLength>`
   `<yourString>` is the string you use for this task and `<stringLength>` is the computed length of that string.