**مدخل إلى الخوارزميات والبرمجة     هندسة الميكاترونيكس سنة أولى**

2023-2024                    مدرس المقرر: د.عيسى الغنام

## Lecture No.8

## Part1 strings

## C++ Strings

Strings are used for storing text.

A string variable contains a collection of characters surrounded by double quotes:

Example

Create a variable of type string and assign it a value:

```
string greeting = "Hello";
```

To use strings, you must include an additional header file in the source code, the <string> library:

Example

```
// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";
```

## String Concatenation

The + operator can be used between strings to add them together to make a new string. This is called **concatenation**:

Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
cout << fullName;
```

In the example above, we added a space after firstName to create a space between John and Doe on output. However, you could also add a space with quotes (" " or ' '):

Example

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + " " + lastName;
cout << fullName;
```

## Append

A string in C++ is actually an object, which contain functions that can perform certain operations on strings. For example, you can also concatenate strings with the append() function:

Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName.append(lastName);
cout << fullName;
```

## C++ Numbers and Strings

## Adding Numbers and Strings

WARNING!

C++ uses the + operator for both **addition** and **concatenation**.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

```
int x = 10;
int y = 20;
int z = x + y;      // z will be 30 (an
integer)
```

If you add two strings, the result will be a string concatenation:

Example

```
string x = "10";
string y = "20";
string z = x + y;   // z will be 1020 (a string)
```

If you try to add a number to a string, an error occurs:

Example

```
string x = "10";
int y = 20;
string z = x + y;
```

## C++ String Length

### String Length

To get the length of a string, use the length() function:

Example

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " <<
txt.length();
```

**Tip:** You might see some C++ programs that use the size() function to get the length of a string. This is just an alias of length(). It is completely up to you if you want to use length() or size():

Example

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "The length of the txt string is: " <<
txt.size();
```

## Access Strings

You can access the characters in a string by referring to its index number inside square brackets [].

This example prints the **first character** in **myString**:

Example

```
string myString = "Hello";
cout << myString[0];
// Outputs H
```

**Note:** String indexes start with 0: [0] is the first character. [1] is the second character, etc.

This example prints the **second character** in **myString**:

Example

```
string myString = "Hello";
cout << myString[1];
// Outputs e
```

## Change String Characters

To change the value of a specific character in a string, refer to the index number, and use single quotes:

Example

```
string myString = "Hello";
myString[0] = 'J';
cout << myString;
// Outputs Jello instead of Hello
```

## C++ Special Characters

## Strings - Special Characters

Because strings must be written within quotes, C++ will misunderstand this string, and generate an error:

```
string txt = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

| Escape character | Result | Description |
|---|---|---|
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |

The sequence \" inserts a double quote in a string:

Example

```
string txt = "We are the so-called \"Vikings\" from the north.";
```

The sequence \' inserts a single quote in a string:

Example

```
string txt = "It\'s alright.";
```

The sequence \\ inserts a single backslash in a string:

Example

```
string txt = "The character \\ is called backslash.";
```

Other popular escape characters in C++ are:

| Escape Character | Result |
|---|---|
| \n | New Line |
| \t | Tab |

## User Input Strings

It is possible to use the extraction operator >> on cin to display a string entered by a user:

Example

```
string firstName;
cout << "Type your first name: ";
cin >> firstName; // get user input from the keyboard
cout << "Your name is: " << firstName;

// Type your first name: John
// Your name is: John
```

However, cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word (even if you type many words):

Example

```
string fullName;
cout << "Type your full name: ";
cin >> fullName;
cout << "Your name is: " << fullName;

// Type your full name: John Doe
// Your name is: John
```

From the example above, you would expect the program to print "John Doe", but it only prints "John".

That's why, when working with strings, we often use the getline() function to read a line of text. It takes cin as the first parameter, and the string variable as second:

Example

```
string fullName;
cout << "Type your full name: ";
getline (cin, fullName);
cout << "Your name is: " << fullName;

// Type your full name: John Doe
// Your name is: John Doe
```

## C++ String Namespace-Omitting Namespace

You might see some C++ programs that runs without the standard namespace library. The using namespace std line can be omitted and replaced with the std keyword, followed by the :: operator for string (and cout) objects:

Example

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

int main() {
  std::string greeting = "Hello";
  std::cout << greeting;
  return 0;
}
```

## C++ References

## Creating References

A reference variable is a "reference" to an existing variable, and it is created with the & operator:

```
string food = "Pizza";  // food variable
string &meal = food;    // reference to food
```

Now, we can use either the variable name food or the reference name meal to refer to the food variable:

Example

```
string food = "Pizza";
string &meal = food;

cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

## C++ Memory Address-Memory Address

In the example from the previous page, the & operator was used to create a reference variable. But it can also be used to get the memory address of a variable; which is the location of where the variable is stored on the computer.

When a variable is created in C++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.

To access it, use the & operator, and the result will represent where the variable is stored:

Example

```
string food = "Pizza";

cout << &food; // Outputs 0x6dfed4
```

**Note:** The memory address is in hexadecimal form (0x..). Note that you may not get the same result in your program.

## C++ Pointers-Creating Pointers

You learned from the previous chapter, that we can get the **memory address** of a variable by using the & operator:

Example

```
string food = "Pizza"; // A food variable of type string

cout << food;  // Outputs the value of food (Pizza)
cout << &food; // Outputs the memory address of food (0x6dfed4)
```

A **pointer** however, is a variable that **stores the memory address as its value**.

A pointer variable points to a data type (like int or string) of the same type, and is created with the * operator. The address of the variable you're working with is assigned to the pointer:

Example

```
string food = "Pizza";  // A food variable of type string
string* ptr = &food;    // A pointer variable, with the name ptr,
that stores the address of food

// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";

// Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";
```

Example explained

Create a pointer variable with the name ptr, that **points to** a string variable, by using the asterisk sign * (string* ptr). Note that the type of the pointer has to match the type of the variable you're working with.

Use the & operator to store the memory address of the variable called food, and assign it to the pointer.

Now, ptr holds the value of food's memory address.

**Tip:** There are three ways to declare pointer variables, but the first way is preferred:

```
string*                       mystring; //                   Preferred
string                                                        *mystring;
string * mystring;
```

## C++ Dereference- Get Memory Address and Value

In the example from the previous page, we used the pointer variable to get the memory address of a variable (used together with the & **reference** operator). However, you can also use the pointer to get the value of the variable, by using the * operator (the **dereference** operator):

Example

```
string food = "Pizza";  // Variable declaration
string* ptr = &food;    // Pointer declaration

// Reference: Output the memory address of food with the pointer
(0x6dfed4)
cout << ptr << "\n";

// Dereference: Output the value of food with the pointer (Pizza)
cout << *ptr << "\n";
```

Note that the * sign can be confusing here, as it does two different things in our code:

When used in declaration (string* ptr), it creates a **pointer variable**.

When not used in declaration, it act as a **dereference operator**.

## C++ Modify Pointers-Modify the Pointer Value

You can also change the pointer's value. But note that this will also change the value of the original variable:

Example

```
string food = "Pizza";
string* ptr = &food;

// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";

// Access the memory address of food and output its value (Pizza)
cout << *ptr << "\n";

// Change the value of the pointer
*ptr = "Hamburger";

// Output the new value of the pointer (Hamburger)
cout << *ptr << "\n";
```

```
// Output the new value of the food variable (Hamburger)
cout << food << "\n";
```

## And why is it useful to know the memory address?

**References** and **Pointers** are important in C++, because they give you the ability to manipulate the data in the computer's memory - **which can reduce the code and improve the performance**.

These two features are one of the things that make C++ stand out from other programming languages, like Python and Java.

انتهت المحاضرة