

Unit 2: The `string` class

Programming 2

Degree in Computer Engineering
University of Alicante
2025-2026



1. The `string` class in C++
2. STL vectors
3. Arguments of a program
4. Exercises

The `string` class in C++

Definition (1/2)

- Character arrays in C can be used in C++, but C++ also has the `string` class* that allows working more easily and flexibly with character strings:

```
// Declaration of a string variable  
string s; // No need to set the string size  
  
// Declaration with initialisation  
string s2="Alicante";  
  
// Declaration of a constant  
const string GREET="hello";
```

*More information on what a “class” is in Unit 5

Definition (2/2)

- A string has a variable size and can dynamically grow depending on the storage needs of the program:

```
string s="hello"; // Stores 5 characters  
s="hello everybody"; // Stores 15 characters*  
s="ok"; // Stores 2 characters
```

- No need to worry about the '`\0`'
- The passing of parameters (value and reference) is done as with any basic data type:

```
void myFunction(string s1, string &s2) {  
    // s1 is passed by value  
    // s2 is passed by reference  
}
```

*A white space counts as any other character

- Screen output with `cout` and `cerr` as with character arrays in C:

```
string s="Mark";  
int num=10;  
  
cout << s << " -> " << num; // Prints "Mark -> 10"
```

Keyboard input > Operator >>

- `cin` and the `>>` operator can be used to read from keyboard in the same way as with character arrays in C
- Blanks before the string are ignored and reading finishes when the first blank is found:

```
string s;  
cin >> s;  
// User writes "    hello"  
// The s variable stores "hello"  
...  
// User writes "good afternoon"  
// The s variable stores "good"
```

Keyboard input > getline (1/2)

- As with character arrays, the function `getline` can be used to read `string` variables
- Reading strings containing blanks is possible in this case:

```
string s;  
getline(cin,s);  
// If the user writes "good afternoon"  
// the variable s stores "good afternoon"
```

- Does not limit the characters read, because with the `string` class is not necessary
- **Alert! The syntax changes with respect to character arrays in C**

Keyboard input > `getline` (2/2)

- If the `>>` operator and `getline` are combined while reading, there is the same problem as with character arrays in C*
- By default, `getline` reads until it finds the newline character (`'\n'`)
- An additional parameter can be passed to indicate that the function must read up to a specific character:

```
string s;  
// Reads until finding the first comma  
getline(cin,s,',');  
// Reads until finding the first square bracket  
getline(cin,s,'[');
```

*The solution is the same as on slide 44 of Unit 1

Extracting words from a string

- Words can be easily extracted from a `string` by using the `stringstream` class:

```
#include <sstream> // Required when using stringstream
...
stringstream ss("Hello cruel world 666");
string s;

// Each iteration of the loop reads until reaching a
blank
while(ss>>s){ // Extracts words one by one
    cout << "Word: " << s << endl;
}
```

string methods (1/3)

- Since `string` is a class, methods are called by putting a dot after the name of the variable
- `length` returns the number of characters in the string:

```
// unsigned int length()  
string s="hello, world";  
cout << s.length(); // Prints 12
```

- `find` returns the position in which a substring appears within a string:

```
// size_t find(const string &s,unsigned int pos=0)  
cout << s.find("world"); // Prints 7  
// If the substring is not found returns string::npos
```

string methods (2/3)

- **replace** substitutes a string (or part of it) with another one:

```
// string& replace(unsigned int pos,unsigned int len,  
    const string &s)  
string s="hello world";  
s.replace(0,5,"hola"); // s is "hola world"
```

- **erase** allows removing part of a string:

```
// string& erase(unsigned int pos=0,unsigned int len=  
    string::npos);  
string s="hello world";  
s.erase(4,3); // s is "hellorld"
```

- **substr** returns a substring of the original string:

```
// string substr(unsigned int pos=0,unsigned int len=  
    string::npos) const;  
string s="hello world";  
string subs=s.substr(2,5); // subs is "llo w"
```

string methods (3/3)

- Example of use:

```
string a="There is a mug in this kitchen with mugs";
string b="mug";
unsigned int size=a.length(); // Length of a
// Search for the first "mug"
size_t found=a.find(b);
if(found!=string::npos){
    cout << "First in: " << found << endl;
    // Search for the second "mug"
    found=a.find(b,found+b.length());
    if(found!=string::npos)
        cout << "Second in: " << found << endl;
}
else{
    cout << "Word '" << b << "' not found";
}
// Replace the first "mug" with "bottle"
a.replace(a.find(b),b.length(),"bottle");
cout << a << endl;
```

Operators (1/2)

- Comparisons: == (equal), != (different), > (greater), >= (greater or equal), < (less) and <= (less or equal)

```
string s1,s2;  
cin >> s1; cin >> s2;  
if(s1==s2) // Comparison in lexicographical order  
    cout << "Equal" << endl;
```

- Assignment of one string to another with the operator =, like any basic data type:

```
string s1="hello";  
string s2;  
s2=s1;
```

- String concatenation with the operator +:

```
string s1="hello";  
string s2="world";  
string s3=s1+", "+s2; // s3 is "hello, world"
```

Operators (2/2)

- Access to components with the operator `[]`, as with character arrays in C:

```
string s="hello";  
char c=s[4]; // s[4] is 'o'  
s[0] = 'H';  
cout << s << ":" << c << endl ; // Prints "Hello:o"
```

- Characters cannot be assigned to positions outside the string:

```
string s;  
s[0]='g'; s[1]='o'; s[2]='o'; s[3]='d';  
// Does not store anything because s is an empty string  
// and these positions are not reserved
```

- Example of traversal of a string character by character:

```
string s="hello, world";  
for(unsigned int i=0;i<s.length(); i++)  
    s[i]='f'; // Replaces each character with 'f'
```

Conversion between string and character array in C

- A character array in C can be assigned to a string using the assignment operator (=):

```
char str[]="hello";  
string s;  
s=str;
```

- A string can be assigned to a character array in C using strcpy and c_str:*

```
char str[10];  
string s="world";  
// There must be enough room in str  
strcpy(str,s.c_str());
```

*The c_str method returns a character array in C with the contents of the string

Conversion between `string` and number

- Transform an integer or real number to `string`:

```
#include <string> // It is not the same as <cstring>  
...  
int num=100;  
string s=to_string(num);
```

- Transform a string to integer:*

```
string s="100";  
int num=stoi(s);
```

- Transform a string to real number:

```
string s="10.5";  
float num=stof(s);
```

*The functions `to_string`, `stoi` and `stof` are available from C++ 2011 version onward

Character array in C vs. string

Character array in C	string
<pre>char str[SIZE]; char str[]="hello"; strlen(str) cin.getline(str, SIZE); if(!strcmp(str1, str2)) {...} strcpy(str1, str2); strcat(str1, str2); strcpy(str, s.c_str());</pre>	<pre>string s; string s="hello"; s.length() getline(cin, s); if(s1==s2){...} s1=s2; s1=s1+s2; s=str;</pre>
Ends with ' <code>\0</code> '	Does not end with ' <code>\0</code> '
Fixed allocated size	Variable allocated size
Variable used size	Used size == allocated size
Can be used with binary files	Cannot be used with binary files

STL vectors

STL vectors (1/3)

- The *Standard Template Library* (STL) is a library of C++ functions
- It provides different data structures and algorithms
- It includes the class `vector`, which allows us to store elements of any type, similarly to regular arrays, but without having to worry about the size:

```
#include <vector> // Include this whenever vector is used
vector<int> vec; // Declares an integer vector
                // Not necessary to indicate its size
```

- The initial size of a STL vector is 0 and it grows dynamically as required
- Use `push_back` to add elements at the end of the vector:*

```
vec.push_back(12); // Adds 12 at the end of the vector
vec.push_back(8); // Adds 8 after 12
```

*Since it is a class, its methods are called by putting a point after the name of the variable

STL vectors (2/3)

- Access to elements by means of the `[]` operator:

```
vec[10]=23; // Similar to regular arrays  
cout << vec[8] << endl;
```

- With `size` we obtain the number of elements of the vector:

```
// Iterate through all elements of the vector  
for(unsigned int i=0;i<vec.size();i++){  
    vec[i]=10;  
}
```

- Range-based vector traversal:

```
// Iterate through all elements of the vector  
for(int num:vec){  
    cout << num << endl;  
}
```

STL vectors (3/3)

- With `clear` we can delete all the elements and with `erase` a specific one:

```
vec.erase(vec.begin()+3); // Deletes the fourth element  
vec.clear(); // Deletes all the elements of the vector
```

- Common error: **you cannot store elements in positions that do not belong to the vector**

```
vector<int> vec;  
vec[0]=78; vec[1]=9; vec[2]=17;  
for(int i=0;i<vec.size();i++){  
    cout << vec[i] << endl; // Prints nothing  
}
```

- There are many other functions for working with STL vectors*

*More information at <http://www.cplusplus.com/reference/vector/vector/>

Arguments of a program

Arguments of a program (1/4)

- The *arguments* of a program are used to provide information (usually options) from the command line
- Their use is very common and allows us to modify the behaviour of the program:

Terminal

```
$ ls          // Lists the files in a directory
$ ls -a       // Lists also hidden files (option "-a")
$ ls -a -l    // Adds extra information from each file (option "-l")
```


Arguments of a program (2/4)

- `main` is a function and as such it can receive two parameters: `argc` and `argv`
- These parameters allow us to manage arguments passed to the program through command line:

```
// Always in this order  
int main(int argc, char *argv[]){  
    ...  
    return 0;  
}
```

- `int argc`: number of arguments passed to the program (including also the program name)
- `char *argv[]`: array of character arrays with the arguments passed to the program

Arguments of a program (3/4)

- Example of use:

```
int main(int argc, char *argv[]) {  
    for(int i=0; i<argc; i++) {  
        cout << "Arg. " << i << " : " << argv[i] << endl;  
    }  
}
```

Terminal

```
$ ./myProgram -a -h X    // Example of call with three parameters  
Arg. 0 : ./myProgram  
Arg. 1 : -a  
Arg. 2 : -h  
Arg. 3 : X
```

- Arguments do not have to start with a hyphen (-) but it is quite a common practice

Arguments of a program (4/4)

- It seems easy to manage arguments, but sometimes things can become complicated
- Users do not always use the same order when introducing arguments:

Terminal

```
$ g++ -Wall -o prog prog.cc -g  
$ g++ -g -Wall prog.cc -o prog
```

- There may be errors in the command-line parameters and help information should be shown to the user
- It is recommended to use a dedicated function to manage the arguments

Exercises

Exercises (1/5)

Exercise 1

Code a function called `subString` that returns a substring of length `n`, starting at position `p` of other string. Both the argument and the return value must be `string` type.

```
subString("heeello",2,5) // Returns "lo"
```

Exercise 2

Code a function `deleteStringCharacter` that, given a `string` and a `character`, deletes all the occurrences of that character in the `string` and returns it.

```
deleteStringCharacter("cocobongo",'o') // Returns "ccbng"
```

Exercises (2/5)

Exercise 3

Code a function `searchSubstring` that searches the first occurrence of a substring `a` inside a string `b` and returns its position, or `-1` if not found. Both `a` and `b` must be `string` type.

```
searchSubstring("eel","heeello") // Returns 2
```

Extensions:

1. Add another parameter to the function that indicates the number of occurrence to return (if the value is `1` it would work as the original function)
2. Implement another function that returns the number of occurrences of the substring in the string

Exercises (3/5)

Exercise 4

Code a function `encrypt` that encodes a string by adding a number `n` to the ASCII code of each character, taking into account that the result must be a character.

For example, if `n=3`, `a` is encoded as `d`, `b` as `e`, ..., `x` as `a`, `y` as `b`, and `z` as `c`.

The function must admit lowercase and uppercase letters. Non-letter characters must not be encoded. The parameter must be `string` type.

```
encrypt("hello, world",3) // Returns "khoor, zruog"
```

Exercises (4/5)

Exercise 5

Write a function `isPalindrome` that returns `true` if the string parameter is a palindrome.

```
isPalindrome("racecar") // Returns true  
isPalindrome("hello, olleh") // Returns false
```

Exercise 6

Implement a function `createPalindrome` that adds to a string the same string but reversed so that the result is a palindrome.

```
createPalindrome("hello") // Returns "helloolleh"
```


Exercises (5/5)

Exercise 7

Implement a program containing a function with the following prototype: `int primeNumber(int n)`. This function will return the n -th prime number. The program must print prime numbers on the screen with the following options:

- `-L` prints each number on a separate line (by default they are all printed on the same line)
- `-N n` prints the n first prime numbers (10 by default)

Execution examples:

Terminal

```
$ primes -N 5
1 2 3 5 7
$ primes -N -L 5
Error: primes [-L] [-N n]
```