Linguagem de Programação II

Prof.Antonio Carlos Sobieranski

DEC7532 | ENC | DEC | CTS

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Unidade 01 – Introdução ao C++

Objetivos da Unidade

a) Breve introdução à Linguagem C++ a partir de conhecimentos prévios da Linguagem C

b) Uso do e-book: **Fundamentals of C++ Programming (Halterman) – Moodle**

O que prestar atenção ao longo dos capítulos 1 a 10:

Cap 1 e 2 – Introduction:

- *output*: *std::cout* (pg.07)

Cap 3. Values and Variables:

- Lista de palavras reservadas do C++ (pg.23)
- Tipos numéricos comuns (a partir pg.24)
- Enumeradores (pg.31 – vamos ver mais tarde)

Cap 4. Expressions and Arithmetic

- *Input*: *std::cin* (pg.37)
- static_cast (pg.42 – vamos ver mais tarde)

Unidade 01 – Introdução ao C++

O que prestar atenção ao longo dos capítulos 1 a 10:

Cap 5. Conditional Execution
- *Boolean type: bool* (pg.85) – *Linguagem C não tem nativo*

Cap 6. Iteration (idem Linguagem C)

Cap 7. Other Conditional and Iterative Statements (idem Linguagem C)

Cap 8. Using Functions (idem Linguagem C)

Cap 9. Writing Functions (idem Linguagem C)

Cap 10. Managing Functions and Data
- *Static Variables: bool* (pg.249) (visto em P1 ?)
- Function overload (pg.251) (visto em P1 ?)
- Recursion (pg.254) (visto em P1 ?)
- Pointers2Functions (pg.277) (visto em P1 ?)

Nossa disciplina inicia a partir daqui (Cap 11 e 13) + alguns aspectos dos Cap. anteriores.

Cap 11. Sequences

Cap 12. Sorting and Searching (visto em outra disciplina, ignorar)

Cap 13. Standard C++ Classes

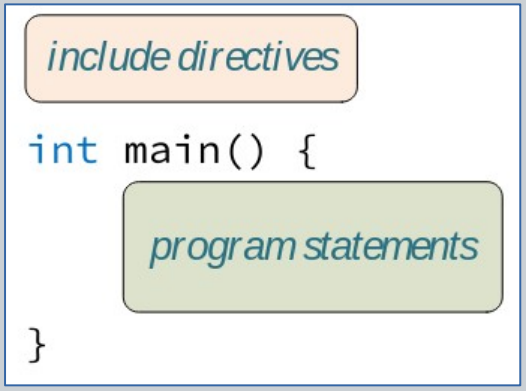
Unidade 01 – Introdução ao C++

Parte 1 – *Basics*: Palavras reservadas em C++

alignas	decltype	namespace	struct
alignof	default	new	switch
and	delete	noexcept	template
and_eq	double	not	this
asm	do	not_eq	thread_local
auto	dynamic_cast	nullptr	throw
bitand	else	operator	true
bitor	enum	or	try
bool	explicit	or_eq	typedef
break	export	private	typeid
case	extern	protected	typename
catch	false	public	union
char	float	register	unsigned
char16_t	for	reinterpret_cast	using
char32_t	friend	return	virtual
class	goto	short	void
compl	if	signed	volatile
const	inline	sizeof	wchar_t
constexpr	int	static	while
const_cast	long	static_assert	xor
continue	mutable	static_cast	xor_eq

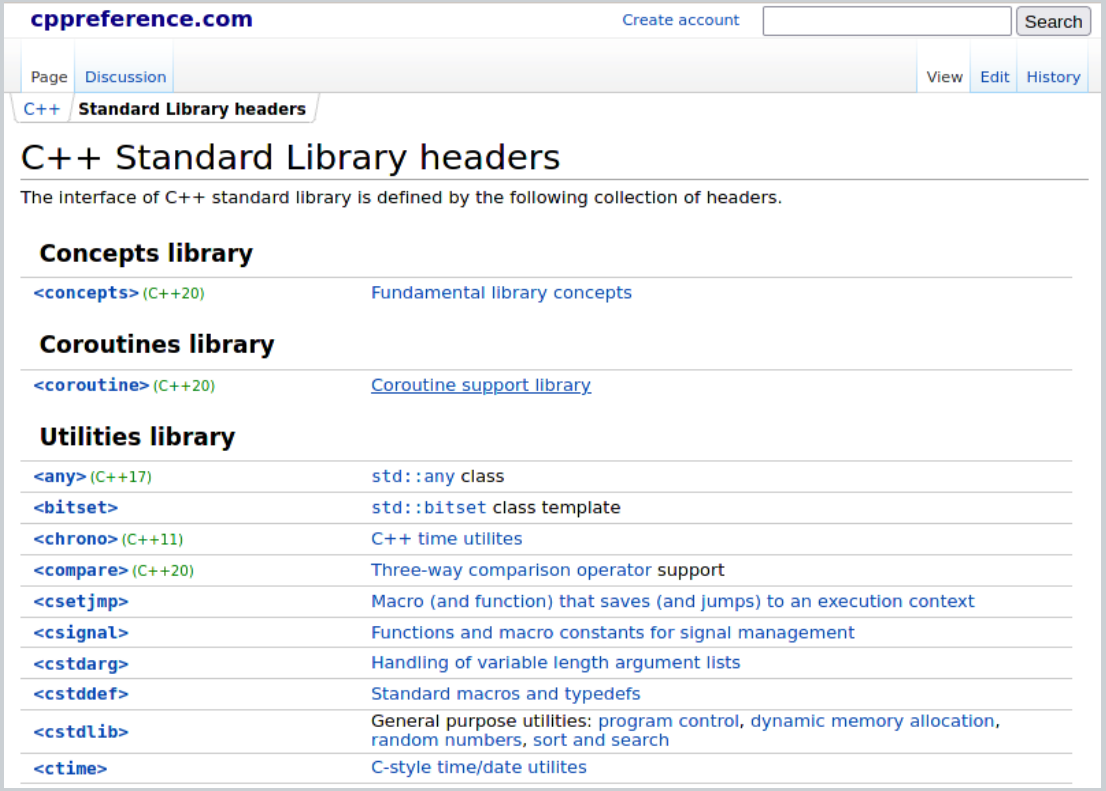
Unidade 01 – Introdução ao C++

Parte 1 – *Basics*: Estrutura Geral de um Programa em C++



Unidade 01 – Introdução ao C++

Parte 1 – *Basics*: Biblioteca padrão do sistema



cppreference.com Create account Search

Page Discussion View Edit History

C++ Standard Library headers

C++ Standard Library headers

The interface of C++ standard library is defined by the following collection of headers.

Concepts library

<concepts> (C++20)	Fundamental library concepts

Coroutines library

<coroutine> (C++20)	Coroutine support library

Utilities library

<any> (C++17)	`std::any` class
<bitset>	`std::bitset` class template
<chrono> (C++11)	C++ time utilites
<compare> (C++20)	Three-way comparison operator support
<csetjmp>	Macro (and function) that saves (and jumps) to an execution context
<csignal>	Functions and macro constants for signal management
<cstdarg>	Handling of variable length argument lists
<cstddef>	Standard macros and typedefs
<cstdlib>	General purpose utilities: program control, dynamic memory allocation, random numbers, sort and search
<ctime>	C-style time/date utilites

https://en.cppreference.com/w/cpp/header

Unidade 01 – Introdução ao C++

Parte 1 – *Basics*: Biblioteca padrão do sistema

Input/output library	
<cstdio>	C-style input-output functions
<fstream>	std::basic_fstream, std::basic_ifstream, std::basic_ofstream class templates and several typedefs
<iomanip>	Helper functions to control the format of input and output
<ios>	std::ios_base class, std::basic_ios class template and several typedefs
<iosfwd>	Forward declarations of all classes in the input/output library
<iostream>	Several standard stream objects
<istream>	std::basic_istream class template and several typedefs
<ostream>	std::basic_ostream, std::basic_iostream class templates and several typedefs
<spanstream> (C++23)	std::basic_spanstream, std::basic_ispanstream, std::basic_ospanstream class templates and typedefs
<sstream>	std::basic_stringstream, std::basic_istringstream, std::basic_ostringstream class templates and several typedefs
<streambuf>	std::basic_streambuf class template
<strstream> (deprecated in C++98)	std::strstream, std::istrstream, std::ostrstream
<syncstream> (C++20)	std::basic_osyncstream, std::basic_syncbuf, and typedefs

https://en.cppreference.com/w/cpp/header

Unidade 01 – Introdução ao C++

Parte 1 – *Basics*: Primeiro Programa em C++

Utilizar alguma IDE de desenvolvimento em C++, editar, compilar e executar

Listing 2.1: simple.cpp

```
#include <iostream>

int main() {
    std::cout << "This is a simple C++ program!\n";
}
```

preprocessing directive

Insertion operator

```
This is a simple C++ program!
```

Ou qualquer editor de texto, e compilar manualmente com o comando:
g++ simple.cpp -o exec

Unidade 01 – Introdução ao C++

Parte 1 – *Basics*: Primeiro Programa em C++

Listing 2.2: simple2.cpp

```
#include <iostream>

using std::cout;

int main() {
    cout << "This is a simple C++ program!\n";
}
```

Listing 2.3: simple3.cpp

```
#include <iostream>

using namespace std;

int main() {
    cout << "This is a simple C++ program!\n";
}
```

Unidade 01 – Introdução ao C++

Printf → ***std::cout*** **<<**
Scanf → ***std::cin*** **>>**

Parte 1 – *Basics*: Input e Output em C++

Saída de uma variável inteiro com ***std::cout***

Listing 3.5: multipleassignment.cpp

```
#include <iostream>

int main() {
    int x;
    x = 10;
    std::cout << x << '\n';
    x = 20;
    std::cout << x << '\n';
    x = 30;
    std::cout << x << '\n';
}
```

Em C++, **std::endl** pode ser utilizado para a quebra de linha.

Unidade 01 – Introdução ao C++

Printf → ***std::cout*** ***<<***
Scanf → ***std::cin*** ***>>***

Parte 1 – *Basics*: Input e Output em C++

Entrada de 02 variáveis inteiras com ***std::cin***
Armazenamento da soma em ***sum***

Listing 4.1: adder.cpp

```
#include <iostream>

int main() {
    int value1, value2, sum;
    std::cout << "Please enter two integer values: ";
    std::cin >> value1 >> value2;
    sum = value1 + value2;
    std::cout << value1 << " + " << value2 << " = " << sum << '\n';
}
```

Unidade 01 – Introdução ao C++

Parte 2. Novos elementos do C++ (Cap.11 e 13):

std::string
std::vector
std::ifstream
std::ofstream

Unidade 01 – Introdução ao C++

Parte 2.a. *String* (Cap.13):

A string is a sequence of characters, most often used to represent words and names. The C++ standard library provides the class `string` which specifies *string objects*. In order to use string objects, you must provide the preprocessor directive

```
#include <string>
```

The `string` class is part of the standard namespace, which means its full type name is `std::string`. If you use the

```
using namespace std;
```

or

```
using std::string;
```

statements in your code, you can use the abbreviated name `string`.

You declare a `string` object like any other variable:

```
string name;
```

```
string name = "joe";
std::cout << name << '\n';
name = "jane";
std::cout << name << '\n';
```

Unidade 01 – Introdução ao C++

Parte 2.a. *String* – Métodos:

- `operator[]`—provides access to the value stored at a given index within the string
- `operator=`—assigns one string to another
- `operator+=`—appends a string or single character to the end of a `string` object
- `at`—provides bounds-checking access to the character stored at a given index
- `length`—returns the number of characters that make up the string
- `size`—returns the number of characters that make up the string (same as `length`)
- `find`—locates the index of a substring within a `string` object
- `substr`—returns a new `string` object made of a substring of an existing `string` object
- `empty`—returns true if the string contains no characters; returns false if the string contains one or more characters
- `clear`—removes all the characters from a string

```
string word = "computer";
std::cout << "\"" << word << "\" contains " << word.length()
          << " letters." << '\n';
```

prints

```
"computer" contains 8 letters.
```

Unidade 01 – Introdução ao C++

Parte 2.a. *String* – Métodos:

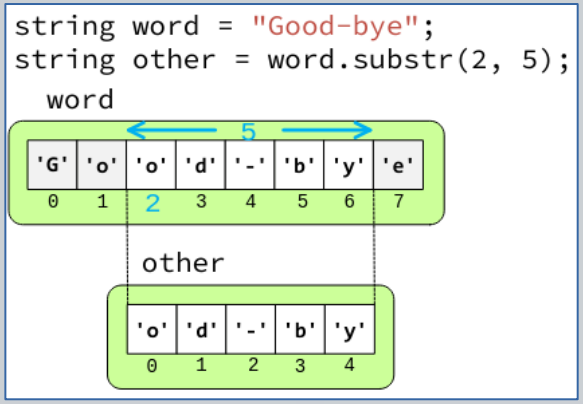
Listing 13.1: stringoperations.cpp

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

int main() {
    // Declare a string object and initialize it
    std::string word = "fred";
    // Prints 4, since word contains four characters
    std::cout << word.length() << '\n';
    // Prints "not empty", since word is not empty
    if (word.empty())
        std::cout << "empty\n";
    else
        std::cout << "not empty\n";
    // Makes word empty
    word.clear();
    // Prints "empty", since word now is empty
    if (word.empty())
        std::cout << "empty\n";
    else
        std::cout << "not empty\n";
    // Assign a string using operator= method
    word = "good";
    // Prints "good"
    std::cout << word << '\n';
    // Append another string using operator+= method
    word += "-bye";
    // Prints "good-bye"
    std::cout << word << '\n';
    // Print first character using operator[] method
    std::cout << word[0] << '\n';
    // Print last character
    std::cout << word[word.length() - 1] << '\n';
```

```
// Prints "od-by", the substring starting at index 2 of length 5
std::cout << word.substr(2, 5);
std::string first = "ABC", last = "XYZ";
// Splice two strings with + operator
std::cout << first + last << '\n';
std::cout << "Compare " << first << " and ABC: ";
if (first == "ABC")
    std::cout << "equal\n";
else
    std::cout << "not equal\n";
std::cout << "Compare " << first << " and XYZ: ";
```

```
    if (first == "XYZ")
        std::cout << "equal\n";
    else
        std::cout << "not equal\n";
}
```



Unidade 01 – Introdução ao C++

Parte 2.b. Vetores (Cap.11):

A vector in C++ is an object that manages a block of memory that can hold multiple values simultaneously; a vector, therefore, represents a collection of values. A vector has a name, and we may access the values it contains via their position within the block of memory managed by the vector. A vector stores a sequence of values, and the values must all be of the same type. A collection of values all of the same type is said to be *homogeneous*.

In order to use a vector object within a C++ program, you must add the preprocessor directive

```
#include <vector>
```

The `vector` type is part of the standard (`std`) namespace, so its full name is `std::vector`, just like the full name of `cout` is `std::cout` (see Section 2.1). If you include the directive

```
using std::vector;
```

in your source file, you can use the shorter name `vector` within your code.

We may declare a vector object that can hold integers as simply as

```
std::vector<int> vec_a;
```

Unidade 01 – Introdução ao C++

Parte 2.b. Vetores:

Declaração e inicialização:

We may declare a vector object that can hold integers as simply as

```
std::vector<int> vec_a;
```

We can declare a vector with a particular initial size as follows:

```
std::vector<int> vec_b(10);
```

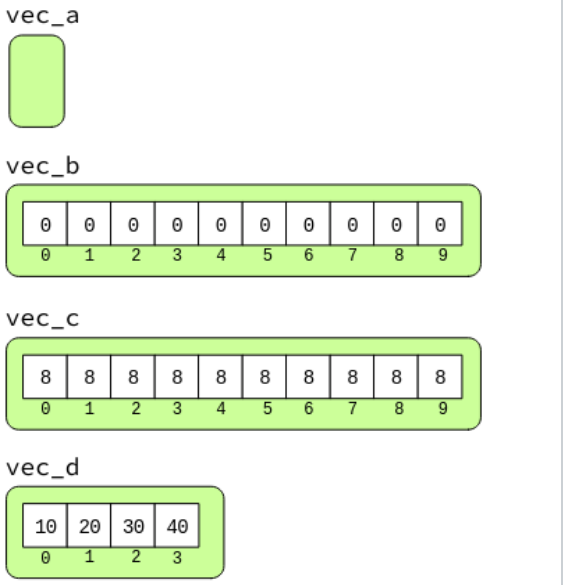
We may declare a vector object of a given size and specify the initial value of all of its elements:

```
std::vector<int> vec_c(10, 8);
```

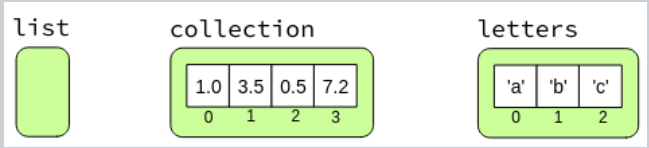
We may declare a vector and specify each and every element separately:

```
std::vector<int> vec_d{10, 20, 30, 40, 50};
```

```
vector<int> vec_a;
vector<int> vec_b(10);
vector<int> vec_c(10, 8);
vector<int> vec_d{ 10, 20, 30, 40 };
```




```
std::vector<int> list;
std::vector<double> collection{ 1.0, 3.5, 0.5, 7.2 };
std::vector<char> letters{ 'a', 'b', 'c' };
```



Unidade 01 – Introdução ao C++

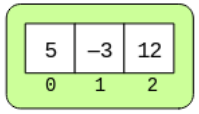
Parte 2.b. Vetores:

Acesso e atribuição:

```
std::vector<int> list(3);  // Declare list to be a vector of three ints
list[0] = 5 ;              // Make the first element 5
list[1] = -3 ;             // Make the second element -3
list[2] = 12 ;             // Make the last element 12
std::cout << list[1] << '\n'; // Print the element at index 1
```

```
vector<int> list(3);
list[0] = 5;
list[1] = -3;
list[2] = 12;
```

list




```
std::vector<int> set(1000);
for (int i = 0; i < 1000; i++)
    set[i] = i;
```

Unidade 01 – Introdução ao C++

Parte 2.b. Vetores – Métodos:

Vectors support a number of methods, but we will focus on seven of them:

- `push_back`—inserts a new element onto the back of a vector
- `pop_back`—removes the last element from a vector
- `operator[]`—provides access to the value stored at a given index within the vector
- `at`—provides bounds-checking access to the value stored at a given position within the vector
- `size`—returns the number of values currently stored in the vector
- `empty`—returns true if the vector contains no elements; returns false if the vector contains one or more elements
- `clear`—makes the vector empty.

```
std::vector<int> list;   // Declare list to be a vector
list.push_back(5);       // Add 5 to the end of list
list.push_back(-3);      // Add -3 to the end of the list
list.push_back(12);      // Add 12 to the end of list
list.pop_back();         // Removes 12 from the list
list.pop_back();         // Removes -3 from the list
```

Unidade 01 – Introdução ao C++

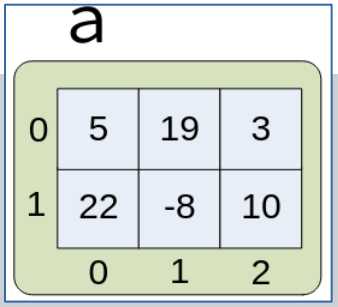
Parte 2.b. Vetores Multidimensionais:

The vectors we have seen thus far have been one dimensional—simple sequences of values. C++ supports higher-dimensional vectors. A *two-dimensional vector* is best visualized as a table with rows and columns. The statement

```
std::vector<std::vector<int>> a(2, std::vector<int>(3));
```

effectively declares `a` to be a two-dimensional (2D) vector of integers. It literally creates a vector with two elements, and each element is itself a vector containing three integers. Note that the type of `a` is a vector of vector of integers. A 2D vector is sometimes called a *matrix*. In this case, the declaration specifies that 2D vector `a` contains two rows and three columns. Figure 11.5 shows the logical structure of the vector created by the following sequence of code:

```
std::vector<std::vector<int>> a(2, std::vector<int>(3));
a[0][0] =  5;
a[0][1] = 19;
a[0][2] =  3;
a[1][0] = 22;
a[1][1] = -8;
a[1][2] = 10;
```




```
std::vector<std::vector<int>> a{{ 5, 19,  3},
                                {22, -8, 10}};
```

Unidade 01 – Introdução ao C++

Parte 2.b. Vetores X Arrays *(std::vector<int> X int a[1000])*

Vector

→ objeto, possui métodos
→ não-contíguo em memória
→ altamente escálavel e auto-gerenciável

Array

→ primitiva
→ contíguo em memória
→ pouco escálavel e gerenciável por alocação de memória (se *dynamic*)

Unidade 01 – Introdução ao C++

Exercício

Criar um programa que realiza a leitura via *std::cin* de palavras (sem espaço, com *std::string, std::vector*). Elaborar um menu contendo as seguintes opções:

UFxC String Store V.0

1. Insert string
2. Print index and string
3. Search string (literal)
4. Search substrings
5. Remove string (by index)
6. Remove by substrings (all occurrences)

0. Quit

Contato

Prof.Antonio Carlos Sobieranski – DEC | A316JD / 206MA

E-mail: a.sobieranski@ufsc.br

https://lsim.ufsc.br

UNIVERSIDADE FEDERAL
DE SANTA CATARINA