

# Zmienne i typy -

<http://www.cplusplus.com/doc/tutorial/variables/>

## ● Identifiers

A *valid identifier* is a sequence of one or more letters, digits, or underscore characters (`_`). Spaces, punctuation marks, and symbols cannot be part of an identifier. In addition, identifiers shall always begin with a letter. They can also begin with an underline character (`_`), but such identifiers are -on most cases- considered reserved for compiler-specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case can they begin with a digit.

C++ uses a number of keywords to identify operations and data descriptions; therefore, identifiers created by a programmer cannot match these keywords. The standard reserved keywords that cannot be used for programmer created identifiers are:

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

Specific compilers may also have additional specific reserved keywords.

**Very important:** The C++ language is a "case sensitive" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the `RESULT` variable is not the same as the `result` variable or the `Result` variable. These are three different identifiers identifying three different variables.

# Zmienne i typy -

<http://www.cplusplus.com/doc/tutorial/variables/>

Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:

- **Character types:** They can represent a single character, such as 'A' or '\$'. The most basic type is `char`, which is a one-byte character. Other types are also provided for wider characters.
- **Numerical integer types:** They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.
- **Floating-point types:** They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
- **Boolean type:** The boolean type, known in C++ as `bool`, can only represent one of two states, true or false.

# Zmienne i typy -

<http://www.cplusplus.com/doc/tutorial/variables/>

Here is the complete list of fundamental types in C++:

Group	Type names*	Notes on size / precision
Character types	<code>char</code>	Exactly one byte in size. At least 8 bits.
	<code>char16_t</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>char32_t</code>	Not smaller than <code>char16_t</code> . At least 32 bits.
	<code>wchar_t</code>	Can represent the largest supported character set.
Integer types (signed)	<code>signed char</code>	Same size as <code>char</code> . At least 8 bits.
	<code>signed short int</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>signed int</code>	Not smaller than <code>short</code> . At least 16 bits.
	<code>signed long int</code>	Not smaller than <code>int</code> . At least 32 bits.
	<code>signed long long int</code>	Not smaller than <code>long</code> . At least 64 bits.
Integer types (unsigned)	<code>unsigned char</code>	(same size as their signed counterparts)
	<code>unsigned short int</code>	
	<code>unsigned int</code>	
	<code>unsigned long int</code>	
	<code>unsigned long long int</code>	
Floating-point types	<code>float</code>	
	<code>double</code>	Precision not less than <code>float</code>
	<code>long double</code>	Precision not less than <code>double</code>
Boolean type	<code>bool</code>	
Void type	<code>void</code>	no storage
Null pointer	<code>decltype(nullptr)</code>	

# Zmienne i typy -

<http://www.cplusplus.com/doc/tutorial/variables/>

Size	Unique representable values	Notes
8-bit	256	= $2^8$
16-bit	65 536	= $2^{16}$
32-bit	4 294 967 296	= $2^{32}$ (~4 billion)
64-bit	18 446 744 073 709 551 616	= $2^{64}$ (~18 billion billion)

For integer types, having more representable values means that the range of values they can represent is greater; for example, a 16-bit unsigned integer would be able to represent 65536 distinct values in the range 0 to 65535, while its signed counterpart would be able to represent, on most cases, values between -32768 and 32767. Note that the range of positive values is approximately halved in signed types compared to unsigned types, due to the fact that one of the 16 bits is used for the sign; this is a relatively modest difference in range, and seldom justifies the use of unsigned types based purely on the range of positive values they can represent.

For floating-point types, the size affects their precision, by having more or less bits for their significant and exponent.

If the size or precision of the type is not a concern, then `char`, `int`, and `double` are typically selected to represent characters, integers, and floating-point values, respectively. The other types in their respective groups are only used in very particular cases.

The properties of fundamental types in a particular system and compiler implementation can be obtained by using the `numeric_limits` classes (see standard header `<limits>`). If for some reason, types of specific sizes are needed, the library defines certain fixed-size type aliases in header `<cstdint>`.