**activation record** - a data structure, that holds information about function, that is being executed

**argument** - a value that we pass to the function

**argument passing** - providing to function a set of arguments

**call stack** - an order in which functions are being executed. Works according to “last in, first out” rule

**class scope** - a text of area within a class

**const** - a keyword used to define constant variables, that in most cases can’t be updated with the new value once defined

**constexpr** - functions that are executed during the compile time

**declaration** - stating that somewhere in program exists such an object

**definition** - providing the compiler enough info about an object, in order the compiler allocates memory for it

**extern** - a keyword used to declare variables. States that the definition of variable is in another place

**forward declaration**  - a declaration of an entity for which the whole definition has not been given yet. For example - two functions call each other and if we won’t declare one of them before another, the first won’t be able to call the second one.

**function** - named part of a program, that can take a set of arguments and return a value

**function definition** - introducing of what parameters does the function require, what is the type of them and providing the code that has to be executed, once the function is called

**global scope** - the are of text outside any other scope

**header file** - a file containing declarations used to share interfaces between the parts of a program

**initializer** - an initial value of an object

**local scope** - a scope inside { } and within parameters block of function

**namespace** - a named space of the program that has no type

**namespace scope** - a named scope nested in the global scope or another namespace

**nested block** - blocks within functions and other blocks

**parameter** - a declaration if an explicit input, passed to a function or a template. When called, function can access the argument through the name of its parameter

**pass-by-const-reference** - passing a value to the function. Function doesn’t create an additional variable to hold the value and uses referenced value directly, but can’t update the value of it.

**pass-by-reference** - passing a value to the function so that function uses it directly and can update the value of it.

**pass-by-value** - when the function is called, the value of an argument is copied to another variable, which is created once the function called and destroyed upon the end of the function

**recursion** - when a function call itself

**return** - terminates the function and in most cases provides an value

**return value** - a value that is being returned by the function

**scope** - a region of program text

**statement scope**  - a scope in for, while, if, switch blocks

**technicalities** - technical aspects

**undeclared identifier** - an object that hasn’t been introduced to the program yet

**using declaration** - used to refer to some particular object of given namespace

**using directive** - states that from this moment and till the end of the scope the objects from given namespace will be used