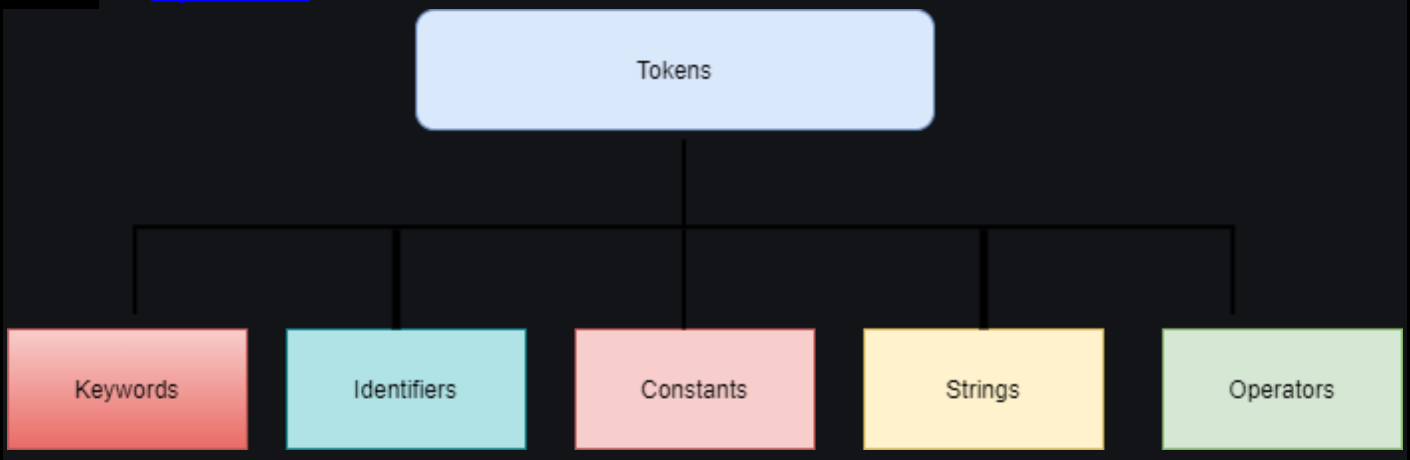
C++ Keywords

C++ is a powerful language. In C++, we can write structured programs and object-oriented programs also. C++ is a superset of C and therefore most constructs of C are legal in C++ with their meaning unchanged. However, there are some exceptions and additions.

Token

When the compiler is processing the source code of a C++ program, each group of characters separated by white space is called a token. Tokens are the smallest individual units in a program. A C++ program is written using tokens. It has the following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators



Keywords

Keywords(also known as **reserved words**) have special meanings to the C++ compiler and are always written or typed in short(lower) cases. Keywords are words that the language uses for a special purpose, such as **void**, **int**, **public**, etc. It can't be used for a variable name or function name or any other identifiers. The total count of reserved keywords is 95. Below is the table for some commonly used C++ keywords.

C++ Keyword			
asm	double	new	switch
auto	else	operator	template
break	enum	private	this

C++ Keyword			
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

Note: *The keywords not found in ANSI C are shown here in boldface.*

- **asm**: To declare that a block of code is to be passed to the assembler.
- **auto**: A storage class specifier that is used to define objects in a block.
- **break**: Terminates a switch statement or a loop.
- **case**: Used specifically within a switch statement to specify a match for the statement's expression.
- **catch**: Specifies actions taken when an exception occurs.
- **char**: Fundamental data type that defines character objects.
- **class**: To declare a user-defined type that encapsulates data members and operations or member functions.
- **const**: To define objects whose value will not alter throughout the lifetime of program execution.
- **continue**:- Transfers control to the start of a loop.
- **default**:- Handles expression values in a switch statement that are not handled by case.
- **delete**: Memory deallocation operator.

- **do**: indicate the start of a do-while statement in which the sub-statement is executed repeatedly until the value of the expression is logical-false.
- **double:** Fundamental data type used to define a floating-point number.
- **else:** Used specifically in an if-else statement.
- **enum:** To declare a user-defined enumeration data type.
- **extern**: An identifier specified as an extern has an external linkage to the block.
- **float**:- Fundamental data type used to define a floating-point number.
- **for**: Indicates the start of a statement to achieve repetitive control.
- **friend:** A class or operation whose implementation can access the private data members of a class.
- **goto**: Transfer control to a specified label.
- **if**: Indicate the start of an if statement to achieve selective control.
- **inline:** A function specifier that indicates to the compiler that inline substitution of the function body is to be preferred to the usual function call implementation.
- **int:** Fundamental data type used to define integer objects.
- **long:** A data type modifier that defines a 32-bit int or an extended double.
- **new**: Memory allocation operator.
- **operator:** Overloads a c++ operator with a new declaration.
- **private**: Declares class members which are not visible outside the class.
- **protected:** Declares class members which are private except to derived classes
- **public:** Declares class members who are visible outside the class.
- **register:** A storage class specifier that is an auto specifier, but which also indicates to the compiler that an object will be frequently used and should therefore be kept in a register.
- **return**: Returns an object to a function's caller.
- **short:** A data type modifier that defines a 16-bit int number.
- **signed:** A data type modifier that indicates an object's sign is to be stored in the high-order bit.
- **sizeof:** Returns the size of an object in bytes.
- **static:** The lifetime of an object-defined static exists throughout the lifetime of program execution.
- **struct:** To declare new types that encapsulate both data and member functions.
- **switch**: This keyword is used in the "Switch statement".
- **template**: parameterized or generic type.
- **this**: A class pointer points to an object or instance of the class.
- **throw:** Generate an exception.
- **try**: Indicates the start of a block of exception handlers.
- **typedef**: Synonym for another integral or user-defined type.
- **union:** Similar to a structure, struct, in that it can hold different types of data, but a union can hold only one of its members at a given time.

- **unsigned:** A data type modifier that indicates the high-order bit is to be used for an object.
- **virtual**: A function specifier that declares a member function of a class that will be redefined by a derived class.
- **void:** Absent of a type or function parameter list.
- **volatile**: Define an object which may vary in value in a way that is undetectable to the compiler.
- **while**: Start of a while statement and end of a do-while statement.

What is the identifier?

Identifiers refer to the name of variables, functions, arrays, classes, etc. created by the programmer. They are the fundamental requirement of any language.

Rules for naming identifiers:

- Identifier names can not start with a digit or any special character.
- A keyword cannot be used as an identifier name.
- Only alphabetic characters, digits, and underscores are permitted.
- The upper case and lower case letters are distinct. i.e., A and a are different in C++.
- The valid identifiers are GFG, gfg, and geeks_for_geeks.

Examples of good and bad identifiers

Invalid Identifier	Bad Identifier	Good Identifier
Cash prize	C_prize	cashprize
catch	catch_1	catch1
1list	list_1	list1

Example:

- C++

```
// C++ program to illustrate the use
// of identifiers

#include <iostream>
using namespace std;

// Driver Code
```

```cpp
int main()
{
    // Use of Underscore (_) symbol
    // in variable declaration
    int geeks_for_geeks = 1;

    cout << "Identifier result is: "
         << geeks_for_geeks;

    return 0;
}
```

Output

```
Identifier result is: 1
```

How keywords are different from identifiers?

So there are some main properties of keywords that distinguish keywords from identifiers:

Keywords	Identifiers
Keywords are predefined/reserved words	identifiers are the values used to define different programming items like a variable, integers, structures, and unions.
Keywords always start in lowercase	identifiers can start with an uppercase letter as well as a lowercase letter.
It defines the type of entity.	It classifies the name of the entity.
A keyword contains only alphabetical characters,	an identifier can consist of alphabetical characters, digits, and underscores.
It should be lowercase.	It can be both upper and lowercase.

Keywords	Identifiers
No special symbols or punctuations are used in keywords and identifiers.	No special symbols or punctuations are used in keywords and identifiers. The only underscore can be used in an identifier.
Keywords: int, char, while, do.	Identifiers: Geeks_for_Geeks, GFG, Gfg1.

Example: Below is the program for how to use different keywords in the program:

- C++

```
// C++ Program to demonstrate keywords
#include <iostream>
using namespace std;

// Driver Code
int main()
{
    // Variable declaration and
    // initialization
    int n = 2;

    // Switch Case Statement
    switch (n) {
    case 1:
        cout << "Computer Network"
             << endl;
        break;
    case 2:
        cout << "C++" << endl;
        break;
    case 3:
        cout << "DBMS" << endl;
        break;
```

```
    case 4:
        cout << "Data Structure"
             << endl;
        break;
    case 5:
        cout << "Operating System"
             << endl;
        break;
    default:
        cout << "Enter Valid number"
             << endl;
    }

    // Return keyword returns an object
    // to a function's caller
    return 0;
}
```

Output

```
C++
```