What is a token?

A C++ program is composed of tokens which are the smallest individual unit. Tokens can be

one of several things, including keywords, identifiers, constants, operators, or punctuation marks.

There are 6 types of tokens in c++:

1. Keyword

2. Identifiers

3. Constants

4. Strings

5. Special symbols

6. Operators

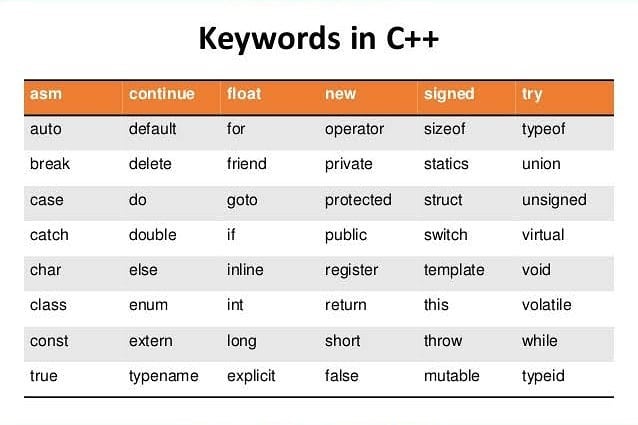
1. Keywords:

In C++, keywords are reserved words that have a specific meaning in the language and

cannot be used as identifiers. Keywords are an essential part of the C++ language and are used

to perform specific tasks or operations.

There are more than 90 keywords in c++:



If you try to use a reserved keyword as an identifier, the compiler will produce an error

because it will not know what to do with the keyword.

Here is an example of a program that tries to use a reserved keyword as an identifier:

#include <iostream>

using namespace std;

int main()

{

// Define a variable named "int" of type int and initialize it to 10

int int = 10;

// Print the value of the "int" variable

cout << "The value of the variable " << int << " is " << int << "." << endl;

return 0;

}

Output:

Explanation:

When we compile the above code, the compiler will produce an error. This error occurs because the

int keyword cannot be used as an identifier. It would help if you chose a different name for your

variable to avoid this error.

In summary, reserved keywords are an essential part of the C++ language and cannot be used

as identifiers. You must choose a different name for your variables, functions, and other objects in

your code.

2. Identifiers

In C++, an identifier is a name given to a variable, function, or another object in the code. Identifiers

are used to refer to these entities in the program, and they can consist of letters, digits, and

underscores. However, some rules must be followed when choosing an identifier:

o The first character must be a letter or an underscore.

o Identifiers cannot be the same as a keyword.

o Identifiers cannot contain any spaces or special characters except for the underscore.

o Identifiers are case-sensitive, meaning that a variable named "myVariable" is different from

a variable named "myvariable".

Here are some examples of valid and invalid identifiers in C++:

o Valid:

o my\_variable

o student\_name

o balance\_due

o Invalid:

o my variable (contains a space)

o student# (contains a special character)

o int (same as a keyword)

It's important to choose meaningful and descriptive names for your identifiers to make your code

easier to read and understand.

Example:

#include <iostream>

using namespace std;

int main()

{

// Define a variable named "age" of type int and initialize it to 25

int age = 25;

// Define a variable named "name" of type string and initialize it to "John Doe"

string name = "rahul";

// Print the value of the "age" and "name" variables

cout << "My name is " << name << " and I am " << age << " years old." << endl;

return 0;

}

Output:

Explanation:

In this program, the variable's age and name are identifiers used to store and access the values of

the variables. The identifier age is used to store a person's age, and the identifier name is used to

store the person's name. Notice how the identifiers are chosen to be descriptive and meaningful,

which makes the code easier to read and understand. You can use any valid identifier in your C++

programs if you follow the rules for naming identifiers discussed earlier.

3. Constants

In C++, a constant is a value that cannot be changed during the execution of the program.

Constants often represent fixed values used frequently in the code, such as the value of pi or the

maximum size of an array.

To define a constant in C++, you use the const keyword followed by the data type and the

constant's name and then initialize it with a value. The value of a constant cannot be changed once

it has been defined. Here is an example:

const double PI = 3.14159;

In this example, the constant PI is of type double and is initialized with the value of pi. You can then

use the constant PI in your code wherever you need to use the value of pi.

Using constants instead of regular variables is important when you need to represent fixed values

in your code. This can make your code more readable and maintainable and prevent accidental

changes to the value of the constant.

Example:

#include <iostream>

using namespace std;

int main()

{

// Define a constant named "PI" of type double and initialize it to 3.14159

const double PI = 3.14159;

// Define a variable named "radius" of type double and initialize it to 10.0

double radius = 10.0;

// Calculate the area of a circle with radius 10.0

double area = PI \* radius \* radius;

// Print the area of the circle

cout << "The area of the circle is " << area << "." << endl;

return 0;

}

Output:

Explanation:

In this program, the constant PI is used to represent the value of pi. The value of PI is defined using

the const keyword and is initialized with the value of pi. The constant PI is then used in calculating

the area of a circle, which is multiplied by the radius of the circle squared to determine the area.

Notice how the constant PI is used the same way as a regular variable, but it cannot be changed

once it has been defined. This makes it a perfect choice for representing fixed values used frequently

in the code.

You can use constants in your C++ programs to represent fixed values that cannot be changed.

This can make your code more readable and maintainable and prevent accidental changes to the

value of the constant.

4. String:

In C++, a string is a sequence of characters that represents text. Strings are commonly used in

C++ programs to store and manipulate text data.

To use strings in C++, you must include the <string> header file at the beginning of your program.

This will provide access to the string class, which is used to create and manipulate strings in C++.

Here is an example:

#include <iostream>

#include <string>

using namespace std;

int main()

{

// Define a string variable named "message" and initialize it with a string value

string message = "Hello, world!";

// Print the value of the "message" variable

cout << message << endl;

return 0;

}

Output:

Explanation:

In this example, the string class creates a string variable named message. The message variable is

initialized with the string "Hello, world!" and then printed to the screen using the cout object.

You can use the string class to perform various operations on strings, such as concatenating strings,

comparing strings, searching for substrings, and more. For more information, you can consult the

documentation for the string class.

5. Special symbols:

In C++, several special symbols are used for various purposes. The ampersand (&) is used to

represent the address of a variable, while the tilde (~) is used to represent the bitwise NOT operator.

Some of the most common special symbols include the asterisk (\*), used as the multiplication and

pointer operators.

The pound sign (#) is used to represent the preprocessor directive, a special type of instruction

processed by the preprocessor before the code is compiled. The percent sign (%) is used as the

modulus operator, which is used to find the remainder when one number is divided by another.

The vertical bar (|) is used to represent the bitwise OR operator, while the caret (^) is used to

represent the bitwise XOR operator. The exclamation point (!) is used to represent the logical NOT

operator, which is used to negate a Boolean value.

In addition to these symbols, C++ also has a number of other special characters that are used for a

variety of purposes. For example, the backslash (/) is used to escape special characters, the double

quotes (") are used to enclose string literals, and the single quotes (') are used to enclose character

literals.

Overall, the special symbols in C++ are an important part of the language and are used in a variety

of different contexts to perform a wide range of operations.

Example:

#include <iostream>

using namespace std;

int main() {

int a = 5;

int b = 10;

int c = a \* b; // Use of the asterisk (\*) as the multiplication operator

cout << "The product of " << a << " and " << b << " is " << c << endl;

int \*ptr = &a; // Use of the ampersand (&) to get the address of a variable

cout << "The address of a is " << ptr << endl;

int d = ~a; // Use of the tilde (~) as the bitwise NOT operator

cout << "The bitwise NOT of " << a << " is " << d << endl;

return 0;

}

Output:

Explanation:

In this program, the asterisk (\*) is used as the multiplication operator to calculate the product of

two variables, a and b. The ampersand (&) is used to get the address of the a variable, which is then

printed to the console. Finally, the tilde (~) is used as the bitwise NOT operator to negate the value

of a.

6. Operators:

In C++, operators are special symbols that are used to perform operations on one or more

operands. An operand is a value on which an operator acts. C++ has a wide range of

operators, including arithmetic operators, relational operators, logical operators, and others.

Arithmetic operators are used to perform mathematical operations such as addition, subtraction,

multiplication, and division. Some examples of arithmetic operators include + (used for addition), -

(used for subtraction), \* (used for multiplication), and / (used for division).

Relational operators are used to compare two values and determine their relationship. Some

examples of relational operators include == (used to check for equality), != (used to check for

inequality), > (used to check if a value is greater than another), and < (used to check if a value is

less than another).

Logical operators are used to combine two or more relational expressions and determine their

logical relationship. Some examples of logical operators include && (used for the logical AND

operation), || (used for the logical OR operation), and ! (used for the logical NOT operation).

In addition to these operators, C++ also has a number of other operators that are used for different

purposes. For example, the bitwise operators are used to perform bitwise operations on individual

bits of a value, and the assignment operators are used to assign a value to a variable.

Overall, operators are an essential part of C++ and are used to perform a wide range of operations

in a program.

Example:

#include <iostream>

using namespace std;

int main() {

int a = 5;

int b = 10;

int c = a + b; // Use of the plus (+) operator for addition

cout << "The sum of " << a << " and " << b << " is " << c << endl;

c = a - b; // Use of the minus (-) operator for subtraction

cout << "The difference between " << a << " and " << b << " is " << c << endl;

c = a \* b; // Use of the asterisk (\*) operator for multiplication

cout << "The product of " << a << " and " << b << " is " << c << endl;

c = a / b; // Use of the forward slash (/) operator for division

}