Why Do We Need a Programming Language?

To communicate with our computers and give instructions to our systems, we need a programming language. Computers can only understand binary language, which is written in zeros and ones. Therefore, we need a programming language to convert our instructions into machine code that computers can understand.

C++ is a general-purpose programming language that supports both procedural and object-oriented programming paradigms. It was developed by Bjarne Stroustrup in 1983 as an extension of the C language. C++ is a compiled language, which means that we need a compiler to convert our code into machine code.

History of C++

Bjarne Stroustrup, a Danish computer scientist, developed C++ in 1979 as an extension of the C language. Initially, it was known as "C with Classes," which combined the features of C with object-oriented programming concepts. Stroustrup wanted to create a language that was fast and efficient, like C, but also supported OOP concepts.

In 1983, the language was renamed to C++, which is an increment operator in C language. The latest version of C++ is C++20, which was released in 2020.

Features of C++

- C++ is a general-purpose programming language that can be used for system programming as well as high-level programming.
- It supports both procedural and object-oriented programming paradigms.
- C++ is a compiled language that requires a compiler to convert code into machine code.
- It is a middle-level language, meaning it supports both low-level and high-level programming.

Let's consider an example program to understand the concept of keywords and identifiers:

```
#include <iostream>using namespace std;int main() {
int a = 10;  cout << a << endl;  return 0;}
```

In this program, int is a keyword, and a is an identifier. The program prints the value of a, which is 10. Note that main is a predefined identifier and can be used as a name of a variable without giving an error.

Here is a list of predefined identifiers:

- cout
- main
- iostream
- cin

You can use these as variable names or for creating new functions. However, it is generally not recommended as it can lead to conflicts in your program. For example, if you use "cout" as a variable name and then try to use "cout" from the "std" namespace, it will cause an error.

It is important to note that some words such as "else" are reserved keywords and cannot be used as identifiers.

When using the "std" namespace, it is a good practice to explicitly write "std::cout" instead of using "using namespace std" to avoid any conflicts.

Reserved words such as "and" and "_eq" are not essential as there are corresponding operators available in the ANSI character set.

Data Types in C++ Programming Language

In the previous video, we discussed keywords and identifiers in C++ programming language. In this video, we will discuss data types, which are essential for programming. We will cover what is a data type, why we need it, and the different types of data types. We will also show you

Increment and Decrement Operators

Increment and decrement operators are used to increase or decrease the value of the operand by 1. There are two types of these operators - pre-increment/decrement and post-increment/decrement.

- Pre-increment (++a)
- Post-increment (a++)
- Pre-decrement (--a)
- Post-decrement (a--)

The main difference between pre-increment/decrement and post-increment/decrement is when the operand is modified. Pre-increment/decrement modifies the value of the operand before it is used, whereas post-increment/decrement modifies the value of the operand after it is used.

Arithmetic and Relational Operators in C++

In C++, arithmetic operators include addition, subtraction, multiplication, division, and modulo. Relational operators are used to show the relationship between operands, such as double equal to, not equal to, less than, greater than, less than or equal to, and greater than or equal to.

Logical Operators

Logical operators combine two or more conditions using boolean operands. There are three logical operators: **logical AND** (&&), **logical OR** (||), and **logical NOT** (!).

- Logical AND: returns true if both operands are true.
- Logical OR: returns true if either one or both operands are true.
- Logical NOT: returns true if the operand is false and vice versa.

The cast operator is used to forcefully convert one data type to another data type.

In addition to the discussion of operators, the video promotes a fun event called "Geek Olympics" by Geeks for Geeks, where participants can win rewards and learn new things every day in July.

Operators and Type Casting in C++

C++ has various types of operators such as arithmetic, relational, logical, bitwise, and assignment operators. In addition, there is also a cast operator used to convert one data type to another.

Cast Operator

To use the cast operator, write the data type you want to convert to in parentheses followed by the variable you want to convert. For example, to convert a float variable `a` to an integer, write `(int)a`.

There are two types of casting: implicit and explicit. Implicit casting is done automatically by the compiler, while explicit casting is done manually using the cast operator.

Miscellaneous and Assignment Operators

Aside from the cast operator, there are also miscellaneous operators such as the conditional operator and assignment operators such as `=`, `+=`, and `-=`.

Understanding Operator Precedence and Associativity in C++

In this series on learning C++ programming language, we have covered all types of operators in C++, including arithmetic, relational, logical, bitwise, assignment, and some miscellaneous operators. The next important topic is operator precedence and associativity. Without understanding this, you cannot solve expressions that involve multiple operators. This topic is crucial for solving expressions that involve

Understanding Precedence and Associativity in C++

When it comes to evaluating expressions in C++, it's important to understand the concepts of precedence and associativity. Precedence refers to the order in which operators are evaluated in an expression, while associativity refers to the order in which operators of the same precedence are evaluated. These concepts are important to keep in mind as they can impact the outcome of an expression.

Example 1

Let's take the expression `x++ + y++`. This expression has both postfix and prefix operators. Postfix operators have a higher precedence and left-to-right associativity, while prefix operators have right-to-left associativity. So, in this case, the expression would be evaluated as follows:

- `x++` - original value of 1 is used, then x becomes 2
- `y++` - 2 is used, then y becomes 3
- `++y` - y is now 3, so it's incremented to 4 and used
- `++x` - x is now 3, so it's incremented to 4 and used

So, if we have `x = 3` and `y = 4`, the output would be 17. However, it's important to note that since we're modifying and accessing the same variable in the same expression, the outcome may be undefined and can vary depending on the compiler.

Example 2

Let's consider the expression `y-- + (y = 10)`. This expression has brackets, which have the highest precedence. The expression is evaluated from left to right. So, the expression would be evaluated as follows:

- `y--` - post decrement operator is used, original value of y (3) is used, then y becomes 2

```
cout << "Let's go for coffee in Starbucks!";      } else
{        cout << "Let's go home.";      }      return 0;
```

In the above example, the if statement checks if the person has more than 1000 rupees. If the condition is true, the program will print "Let's go for coffee in Starbucks!". If the condition is false, the program will print "Let's go home."

Understanding the Simple If Block

Here's a flowchart to help explain the simple `if` block:

- The program prompts the user to enter an amount of money they have
- The user enters the amount of money they have, and it is stored in the `money` variable
- If the `money` variable is greater than or equal to 1000, the program will print "Coffee in Starbucks"
- Regardless of the condition, the program will always print "Let's go home" after the `if` block

It's important to note that the condition can be any expression or constant value, but it must ultimately evaluate to a boolean value (either true or false).

If you want to check for equality in the condition, you must use the double equal sign (`==`) instead of the single equal sign (`=`) which is the assignment operator.

If you want to execute multiple statements after the condition is true, you must enclose them in curly brackets to create a block.

If we have 100 rupees, we will have coffee in a normal shop and then leave.

We can use logical and operators to check more than one condition which are dependent on each other. For example, if we go to the supermarket and both apples and oranges are available, we can buy them. We can write all the conditions within a single if statement using logical and operator.

Instead of using nested if, we can write a program using logical and operator. Try to write a program using nested conditions and then try to write the same program using logical and operator.

Control Structures in C++: Switch Case Statement

In this video, we will be discussing the switch case statement, which is a substitute for long if-else statements. The switch case statement allows us to write readable programs with fewer lines of code. We will cover what the switch case statement is, why we use it, how to use it with a program, important points, and some examples.

Important Points

- The switch case statement is evaluated based on a constant or integral value.
- Only integral types are allowed in the switch expression, such as a constant value, variable, or character.
- Each case value must have an integral value, such as a constant value or character associated with an integer value.
- The break statement is used to terminate the switch and exit the statement.

Event: Skill Dependence Days

Before diving into the switch case statement, I want to let you know that Skill Dependence Days is coming up from August 13th to 15th. During these three days, you can get a flat 15% sitewide discount on the entire

- **Do While Loop:** Similar to the While Loop but executes at least once, regardless of the condition.
- **Range Based For Loop:** Used in C++ for iterating through elements of a container like an array or vector.

Loops in C++: For Loop

In this video, we will be discussing the for loop in C++ programming language. We will cover the general syntax of the for loop, its working, and provide examples and programs.

Why Use Loops?

We use loops when we have repetitive statements or iterations in our program. It allows us to execute a statement or set of statements multiple times.

General Syntax of For Loop

The general syntax of a for loop includes three expressions: initialization, termination condition, and update. These expressions are separated by semicolons and followed by the loop body.

```
for (expression1; expression2; expression3) {  // loop body}
```

Example

Let's say we want to print numbers from 1 to 10. Without using a loop, we would have to write repetitive statements. With a for loop, we can eliminate these repetitive statements.

```
int i;for (i = 1; i <= 10; i++) {  std::cout << i << " ";}
```

In this example, we initialize the variable i to 1. The termination condition is i <= 10, meaning the loop will continue until i is no longer

If a semicolon is added at the end of the for loop statement, it will terminate the loop and anything after it will be considered as a separate statement in the program.

Increment and decrement expressions can be added either in the loop body or as separate expressions separated by commas.

If no termination condition is added, the loop will result in an infinite loop.

Example:

```
    for(int i=0, j=0; i<=5, j<3; i++, j++){
cout << i << " " << j << endl;     }
```

- The loop will print values of i and j until j is less than 3.
- The last value of j will be 2, as it increments until it becomes 3 and the loop stops.
- Adding a semicolon after the loop statement will terminate the loop and anything after it will be considered as a separate statement in the program.
- If no termination condition is added, the loop will result in an infinite loop.

While Loops in C++ Programming Language

In this series of learning C++ programming language, we are discussing loops in C++. Till now, we have discussed for loop and different ways of writing while loop and some tricky things you can do with while loop. In this video, we will be discussing the while loop syntax, flowchart, and the working of the while loop with the help of three or four programs. We will be seeing these things practically, and each and every detail about the while loop will be discussed in this video.

User Input Validation

To validate user input, we can use a while loop. For example, we can ask the user to enter a number greater than 6:

```
int n;
cout << "Enter a number greater than 6: ";
cin >> n;
while (n <= 6) {
  cout << "Please enter a number greater than 6: ";
  cin >> n;
}
cout << "Thanks!" << endl;
```

In the above example, if the user enters a number less than or equal to 6, the program will prompt them to enter a new number until a valid number is entered.

Let's discuss some more about the while loop. Here is a question: we have a variable i, and in the while loop, we are writing a condition termination, i=1. We are printing i, and then i++. What output will you get? The answer is that it will print an infinite loop of 1 1 1 1.... To check the equality condition, we have to put double equal to.

Here is another question: we are asking for the value of i, then in the while loop, we are just writing i and printing i and i++. What output will you get? The answer is that it will not be an infinite loop. The range of int is limited, so it will not print an infinite loop.

Understanding While Loops in Programming

While loops are a powerful tool in programming that allow for the repeated execution of a block of code as long as a certain condition is true. When the condition becomes false, the loop exits. Let's take a look at an example: