Chapter Two: Basics of C++

Statements: In programming languages like C++, statements are used to specify a sequence of operations that the computer should perform. A statement is a single line of code that performs a specific task, such as assigning a value to a variable, calling a function, or performing an arithmetic operation.

Examples of statements in C++ include:

- Variable assignment: int x = 5;

Function call: cout << "Hello World!";

- Conditional statement: if (x > 10) { cout << "x is greater than 10"; }

- Looping statement: for (int i = 0; i < 10; i++) { cout << i; }

Each statement in C++ must end with a semicolon (;) to indicate the end of the statement.

Preprocessor Directives:

Preprocessor directives are a set of commands that are processed by the preprocessor before the actual compilation of the code. These directives provide a way to customize the compilation process and modify the behavior of the code. Preprocessor directives start with the '#' symbol and are placed at the beginning of a line in the code.

Some of the commonly used preprocessor directives in C++ include:

#include: This directive is used to include header files in the code. For example, #include <iostream> is used to include the iostream library for input and output operations.

#define: This directive is used to define constants or macros in the code. For example, #define PI 3.14159 defines a constant PI with a value of 3.14159.

C++ Keywords:

Keywords are reserved words that have a specific meaning and purpose in the C++ language. These words cannot be used as variable names or identifiers because they are already used by the language for specific tasks. Keywords are an essential component of the language, as they define the syntax and structure of the code.

Some of the commonly used C++ keywords include:

- int: This keyword is used to define integer variables.

- double: This keyword is used to define floating-point variables.

- if/else: These keywords are used to implement conditional statements in the code.

- for/while/do-while: These keywords are used to implement loops in the code.

- switch/case/default: These keywords are used to implement switch statements in the code.

C++ Identifiers:

Identifiers are names given to variables, functions, classes, and other user-defined elements in the code. Identifiers are used to represent a value or a concept in the code, and they can be composed of letters, numbers, and underscore characters.

In C++, identifiers must follow certain rules, such as:

Identifiers must begin with a letter or an underscore character. Identifiers cannot begin with a number. Identifiers can be composed of letters, numbers, and underscore characters.

Identifiers are case-sensitive.

Examples of identifiers in C++ include:

- Variable names: int num1, double pi, char letter;

- Function names: void printHello(), int calculateSum(int a, int b);

- Class names: class Rectangle, class Employee;

- Object names: Rectangle rect1, Employee emp1;

C++ Comments:

Comments are text annotations that are added to the code to provide information and improve its readability. Comments are ignored by the compiler, and they do not affect the behavior of the code. In C++, there are two types of comments: single-line comments and multi-line comments.

- Single-line comments:

Single-line comments are used to annotate a single line of code. In C++, single-line comments start with two forward slashes (//). Everything that follows the // symbol on that line is considered a comment and is ignored by the compiler.

Ex:- int num1 = 10; // This is a single-line comment

- Multi-line comments:

Multi-line comments, also known as block comments, are used to annotate multiple lines of code. In C++, multi-line comments start with a forward slash and asterisk (/) and end with an asterisk and forward slash (/). Everything that is between the /\* and \*/ symbols is considered a comment and is ignored by the compiler.

Ex:-/\* This is a

multi-line comment \*/

int num1 = 10;

C++ Variables: Variables are used to store data in a program. In C++, variables must be declared before they can be used. A variable declaration specifies the type of data that the variable can hold, as well as a name that is used to refer to the variable in the program. For example, the following code declares three variables of type int:

int num1;

int num2;

int result;

In this code, we declare three variables named num1, num2, and result of type int.

C++ Variable Declaration:

Variable declaration is the process of creating a variable and specifying its type and name. In C++, variables must be declared before they can be used. A variable declaration consists of the following components:

Data type: Specifies the type of data that the variable can hold, such as int, double, char, etc.

Variable name: Specifies the name that is used to refer to the variable in the program.

Optional initializer: Specifies the initial value of the variable. If no initializer is provided, the variable is initialized with a default value (0 for integers, 0.0 for floating-point numbers, and NULL for pointers). For example, the following code declares a variable named num of type int and initializes it with the value 10:

int num = 10;

In this code, we declare a variable named num of type int and initialize it with the value 10.

Assigning values to variables: In C++, you assign a value to a variable using the assignment operator =. For example, you can declare an integer variable called age and assign it the value of 25 like this:

int age = 25;

You can also assign values to variables using arithmetic operations or other expressions. For example:

int x = 10;

int y = 5;

int z = x + y;

This declares three integer variables, x, y, and z, and assigns them values. The value of z is the sum of x and y.

C++ data types and their ranges:

C++ supports several basic data types, each with a different range of values that it can hold. Here are the most commonly used basic data types in C++, along with their ranges:

int: This data type is used to store integers, both positive and negative, within a range of -2147483648 to 2147483647.

float: This data type is used to store floating-point numbers with single precision. It has a range of approximately 1.2e-38 to 3.4e+38.

double: This data type is used to store floating-point numbers with double precision. It has a range of approximately 2.2e-308 to 1.8e+308.

char: This data type is used to store a single character, such as a letter, digit, or symbol, within a range of -128 to 127 or 0 to 255, depending on whether it is signed or unsigned.

bool: This data type is used to store boolean values, either true or false.

short: This data type is used to store integers with a smaller range than int, within a range of -32768 to 32767.

Characters and numbers: In C++, characters and numbers are both represented using different data types. Characters are represented using the char data type. It can hold a single character within a range of -128 to 127 or 0 to 255, depending on whether it is signed or unsigned. You can assign a character to a variable like this:

char ch = 'A';

In this example, we are assigning the character 'A' to the variable ch.

Numbers, on the other hand, can be represented using several different data types depending on their precision and range.

Input/output statements: Input and output statements are used in C++ to communicate with the user and to read or write data.

Input statements:

cin - reads input from the user through the keyboard. Example: cin >> variable;

Output statements:

cout - displays output on the screen. Example: cout << "Hello, world!";

Operation and operation types: In C++, operations are used to manipulate data and perform calculations on variables. Here are some commonly used operators and their types in C++:

Arithmetic operators:

Used to perform mathematical calculations.

Examples: + (addition), - (subtraction), \* (multiplication), / (division), % (modulus)

Relational operators:

Used to compare two values and return a Boolean result (true or false). Examples: < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), == (equal to), != (not equal to)

Logical operators: Used to combine multiple conditions and return a Boolean result. Examples: && (logical AND), || (logical OR), ! (logical NOT)

Assignment operators: Used to assign a value to a variable.

Examples: = (assignment), += (addition assignment), -= (subtraction assignment), \*= (multiplication assignment), /= (division assignment), %= (modulus assignment)

Increment and decrement operators: Used to increment or decrement the value of a variable by 1. Examples: ++ (increment), -- (decrement)

Bitwise operators: Used to perform operations on the binary representation of values.

Examples: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift).

Increment and decrement operator: Increment and decrement operators are used in C++ to modify the value of a variable by increasing or decreasing it by 1. Here are some important things to know about increment and decrement operators:

The increment operator ++ adds 1 to the variable's value. Example: x++;

The decrement operator -- subtracts 1 from the variable's value. Example: y--;

Increment and decrement operators can be used in prefix or postfix form. In prefix form, the operator appears before the variable name (++x), while in postfix form, it appears after the variable name (x++).

When used in postfix form, the value of the variable is returned before it is incremented or decremented. When used in prefix form, the value is returned after it is incremented or decremented.

Example: y = 5; x = y++; // x is assigned 5 and y becomes 6.

Increment and decrement operators can only be used with variables, not with constants or literals.

It is important to note that overusing increment and decrement operators can lead to hard-to-read code and potential bugs. It is generally recommended to use them sparingly and to prioritize code readability and maintainability.

Precedence of operators: Precedence of operators refers to the order in which operators are evaluated in an expression. In C++, operators have different levels of precedence, and some operators have higher precedence than others. Here is a list of the operator precedence levels in C++, from highest to lowest:

Parentheses ( ) - Used to group expressions and to explicitly specify evaluation order.

Increment and decrement operators ++ -- - Prefix and postfix increment and decrement operators.

Unary operators + - ! ~ - Unary plus, unary minus, logical NOT, and bitwise NOT.

Multiplication, division, and modulus \* / % - Multiply, divide, and take the remainder of two operands.

Addition and subtraction + - - Add and subtract two operands.

Type of conversion: there are two types of type conversion: implicit conversion and explicit conversion.

1-Implicit Conversion: Implicit conversion, also known as type coercion, is the automatic conversion of a value from one data type to another by the compiler, without the need for explicit casting or conversion functions. This can occur when a value of one data type is assigned to a variable of another data type, or when two different data types are used in an expression. The goal of implicit conversion is to ensure that data is used appropriately and efficiently in a program, while also minimizing the risk of data loss or errors.

2-Explicit Conversion: Explicit conversion, also known as type casting, is the manual conversion of a value from one data type to another using special conversion functions or casting operators. This type of conversion is used when the programmer needs to convert a value to a specific data type, or when the compiler is unable to perform an implicit conversion.