Review of constructs of C used in C++ : variables, types and type declarations, user defined data types; increment and decrement operators, relational and logical operators; if then else clause; conditional expressions, input and output statement, loops, switch case, arrays, structure, unions, functions, pointers; preprocessor directives.

C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

- int - stores integers (whole numbers), without decimals, such as 123 or -123
- double - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:
type variable = value;

Declare Many Variables

To declare more than one variable of the **same type**, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
```

User defined Data Types in C++

Data types are means to identify the type of data and associated operations of handling it. There are three types of data types:

1. Pre-defined DataTypes
2. Derived Data Types
3. User-defined DataTypes

User-Defined DataTypes:

The data types that are defined by the user are called the derived datatype or user-defined derived data type. These types include:

- Class
- Structure
- Union
- Enumeration
- Typedef defined DataType

C++ divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Increment and Decrement Operator in C++

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one.

Both increment and decrement operator are used on single operand or variable, so it is called as unary operator. Unary operators are having higher priority than the other operators it means unary operators are execute before other operators.

```
++  // increment operator
--  // decrement operator
```

Type of Increment Operator

- pre-increment
- post-increment

pre-increment (++ variable)

In pre-increment first increment the value of variable and then used inside the expression (initialize into another variable).

Type of Decrement Operator

- pre-decrement
- post-decrement

Relational Operators

There are following relational operators supported by C++ language

Assume variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

There are following logical operators supported by C++ language.

Assume variable A holds 1 and variable B holds 0, then −

Operator	Description	Example

&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

C++ Conditions and If Statements

C++ supports the usual logical conditions from mathematics:

- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- Equal to `a == b`
- Not Equal to: `a != b`

You can use these conditions to perform different actions for different decisions.

C++ has the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

Basic Input / Output in C++

C++ comes with libraries which provides us with many ways for performing input and output. In C++ input and output is performed in the form of a sequence of bytes or more commonly known as **streams**.

- **Input Stream:** If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.

- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

Header files available in C++ for Input/Output operations are:

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions to objects like cin, cout, cerr etc.
2. **iomanip**: iomanip stands for input output manipulators. The methods declared in this files are used for manipulating streams. This file contains definitions of setw, setprecision etc.
3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

The two keywords **cout in C++** and **cin in C++** are used very often for printing outputs and taking inputs respectively. T

Standard output stream (cout): Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).

standard input stream (cin): Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard. The extraction operator(>>) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keboard.

C++ programming language provides the following type of loops to handle looping requirements.

Sr.No	Loop Type & Description
1	while loop :Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop: Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop :Like a ‘while’ statement, except that it tests the condition at the end of the loop body.
4	nested loops:You can use one or more loop inside any another ‘while’, ‘for’ or ‘do..while’ loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C++ supports the following control statements.

Sr.No	Control Statement & Description
1	break statement :Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	continue statement :Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	goto statement :Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

C++ Switch Statements

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The break and default keywords are optional

Arrays

An array is a series of elements of the same type placed in contiguous memory locations that can be individually Initializing arrays

By default, regular arrays of *local scope* (for example, those declared within a function) are left uninitialized. This means that none of its elements are set to any particular value; their contents are undetermined at the point the array is declared.
But the elements in an array can be explicitly initialized to specific values when it is declared, by enclosing those initial values in braces {}. For example:

```
int foo [5] = { 16, 2, 77, 40, 12071 };
```

referenced by adding an index to a unique identifier.

Structure in C++ programming

The `struct` keyword defines a structure type followed by an identifier (name of the structure).
Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
   char name[50];
   int age;
   float salary;
};
```

Union in C++

union is a user-defined type that uses same block of memory for every its list member. **Union** may be useful when it is necessary to work with different representation of same binary data. For **example**, you need to store color data as four 8-bit unsigned char numbers.

C++ Functions.

A **function** is a block of code which only runs when it is called. You can pass data, known as parameters, into a **function**. **Functions** are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

C++ provides some pre-defined functions, such as `main()`, which is used to execute code. But you can also create your own functions to perform certain actions.

To create (often referred to as *declare*) a function, specify the name of the function, followed by parentheses **()**:

Syntax

```
void myFunction() {
  // code to be executed
}
```

Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.To call a function, write the function's name followed by two parentheses () and a semicolon ; In the following example, myFunction() is used to print a text (the action), when it is called

Pointers

The variable that stores the address of another variable (like foo in the previous example) is what in **C++** is called a **pointer**. **Pointers** are a very powerful feature of the language that has many uses in lower level programming

Preprocessors directories:

Preprocessor programs provide preprocessors directives which tell the compiler to preprocess the source code before compiling. All of these preprocessor directives begin with a '#' (hash) symbol. This ('#') symbol at the beginning of a statement in a C/C++ program indicates that it is a pre-processor directive. We can place these preprocessor directives anywhere in our program. Examples of some preprocessor directives are: *#include*, *#define*, *#ifndef* etc.

There are 4 main types of preprocessor directives:

1. Macros
2. File Inclusion
3. Conditional Compilation
4. Other directives

Let us now learn about each of these directives in details.

- **Macros**: Macros are a piece of code in a program which is given some name. Whenever this name is encountered by the compiler the compiler replaces the name with the actual piece of code. The '#define' directive is used to define a macro.

.