HELLO CPP - FIRST C++ CODE

```cpp
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello cpp" << endl;
}
```

PROBLEMS OUTPUT TERMINAL ... Code

```
Hello cpp
PS D:\GauravDemo>
```

BASICS OF C++
PRE-PROCESSOR DIRECTIVES

.The # indicates a command for the c++ preprocessor
    -Known as a preprocessor directive
    -Such commands do not end in a semi-colon
.The C++ preprocessor copies the content of the included file into the program
    -Replacing the line containing the directive
    -Such files contain function headers

KEY-POINTS

01 Reserved words

Reserved words are words that gives an instructions to the compiler.
e.g. int and using

02 Comments

Programs should include English explanations that describe what the program is doing . These are called comments.

03 Variabls

- -Variables are used to store data
- -The value stored in the variable can be changed using the assignment operator,=
-

04 Identifires

User created names must only contain

- Letters, both upper and lower case
- Digits(0 to 9)
- The underscore character(_).



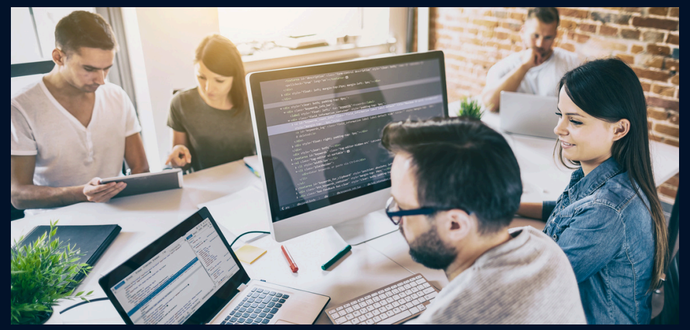

Next Page

BASICS OF C++

STRUCTURE OF C++

- HEADER FILE DECLARATION SECTION
- GLOBAL DECLARATION SECTION
- CLASS DECLARATION AND METHOD DEFINITION
- MAIN FUNCTION
- METHOD DEFINATION SECTION

C++ DATA TYPES



Primary data type

They are fundamental and serve as the building blocks for more complex data types.
int ,float , char, void

User defined data type

user-defined data types allow you to create complex types that are tailored to the specific needs of your program.
structure , union,class ,enumeration

Derived data type

Derived data types in C++ are types that are based on the primary data types or user-defined data types. They include arrays, pointers, references, and classes (with inheritance).

Type	Size (bits)	Size (bytes)	Range
char	8	1	-128 to 127
unsigned char	8	1	0 to 255
int	16	2	-2^{15} to $2^{15}-1$
unsigned int	16	2	0 to $2^{16}-1$
short int	8	1	-128 to 127
unsigned short int	8	1	0 to 255
long int	32	4	-2^{31} to $2^{31}-1$
unsigned long int	32	4	0 to $2^{32}-1$
float	32	4	3.4E-38 to 3.4E+38
double	64	8	1.7E-308 to 1.7E+308
long double	80	10	3.4E-4932 to 1.1E+4932

```cpp
#include <iostream>

int main() {
    int a = 10;                 // Integer
    short b = 20;               // Short integer
    long c = 30000;             // Long integer
    long long d = 123456789;    // Long long integer
    unsigned int e = 40;        // Unsigned integer

    float f = 3.14f;            // Float
    double g = 3.14159;         // Double
    long double h = 3.1415926535; // Long double

    char i = 'A';               // Character
    unsigned char j = 'B';      // Unsigned character
    signed char k = 'C';        // Signed character
    wchar_t l = L'Ω';           // Wide character

    bool m = true;              // Boolean
```

```cpp
    // Void type example is typically used with functions
    void functionExample();

    std::cout << "Integer: " << a << std::endl;
    std::cout << "Short: " << b << std::endl;
    std::cout << "Long: " << c << std::endl;
    std::cout << "Long Long: " << d << std::endl;
    std::cout << "Unsigned Int: " << e << std::endl;
    std::cout << "Float: " << f << std::endl;
    std::cout << "Double: " << g << std::endl;
    std::cout << "Long Double: " << h << std::endl;
    std::cout << "Char: " << i << std::endl;
    std::cout << "Unsigned Char: " << j << std::endl;
    std::cout << "Signed Char: " << k << std::endl;
    std::cout << "Wide Char: " << l << std::endl;
    std::cout << "Bool: " << m << std::endl;

    return 0;
}

void functionExample() {
    std::cout << "This is a void function example." << std::endl;
}
```

OPERATORS IN C++

1. ARITHMETIC OPERATORS
2. RELATIONAL OPERATORS
3. LOGICAL OPERATORS
4. BITWISE OPERATORS
5. ASSIGNMENT OPERATORS
6. TERNARY OR CONDITIONAL OPERATORS

Arithmetic Operators		
Operator	**Description**	**Example**
+	Addition	6 + 2 = 8
-	Subtraction	6 – 2 = 4
*	Multiplication	6 * 2 = 12
/	Division	6 / 2 = 3
%	Modulus	6 % 2 = 0
++	Increment	If a = 6, then a++ gives 7
--	Decrement	If a = 6. then a-- gives 5

Relational Operators

For all examples below consider a = 10 and b = 5

Operator	Description	Example
>	Greater than	a > b gives true
>=	Greater than or equal	a >= b gives false
<	Less than	a < b gives false
<=	Less than or equal	a <= b gives false
= =	Equals	a = = b gives false
! =	Not equals	a ! = b gives true

Logical Operators		
For all examples below consider a = 10 and b = 5		
Operator	**Description**	**Example**
&&	Logical AND	(a>b) && (b==5) gives true
||	Logical OR	(a>b) || (b==2) gives true
!	Logical NOT	!(b==5) gives false

Bitwise Operators

For all examples below consider a = 10 and b = 5

Operator	Description	Example
&	Bitwise AND	a & b gives 0
|	Bitwise OR	a | b gives 15
^	Bitwise Ex-OR	a ^ b gives 15
~	1's complement (NOT)	~a gives some negative value
<<	Left shift	a << 1 gives 20
>>	Right shift	a >> 1 gives 5

Truth Table of Bitwise Operators					
In C++ boolean *true* is 1 and *false* is 0					
a	**b**	**a & b**	**a | b**	**a ^ b**	**~ a**
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Ternary operator