CHARACTER SET IN C++

The character set is a set of words, digits, symbols and operators that are valid in C++.

There are four types of Character Set:-

Character Set		
1.	Letters	Uppercase A-Z Lowercase a-z
2.	Digits	All digits 0-9
3.	Special Characters	All Symbols: , . : ; ? ' " ! | \ / ~ _$ % # & ^ * - + < > ( ) { } [ ]
4.	White Spaces	Blank space, Horizontal tab, Carriage return, New line, Form feed

WHAT ARE TOKENS ??

C++ Tokens are the smallest individual units of a program.

C++ is the superset of C and so most constructs of C are legal in C++ with their meaning and usage unchanged. So tokens, expressions, and data types are similar to that of C.

Following are the C++ tokens : (most of c++ tokens are basically similar to the C tokens)

- Keywords
- Identifiers
- Constants
- Variables
- Operators

Keywords

Keywords are reserved words which have fixed meaning, and its meaning cannot be changed. The meaning and working of these keywords are already known to the compiler. C++ has more numbers of keyword than C, and those extra ones have special working capabilities.

There are 32 of these, and here they are

```
auto const double float int short struct unsigned
break continue elseforlong signed switch void
case default enumgoto register sizeof typedef volatile
char do extern if return static unionwhile
```

There are another 30 reserved words that were not in C, are therefore new to C++, and here they are -

```
asm dynamic_cast namespace reinterpret_cast try
bool explicit new static_cast typeid
catch false operator template typename
class friend privatethis using
const_cast inline public throw virtual
delete mutable protected true wchar_t
```

Identifiers

Identifiers are names given to different entries such as variables, structures, and functions. Also, identifier names should have to be unique because these entities are used in the execution of the program.

Identifier naming conventions

- Only alphabetic characters, digits and underscores are permitted.
- First letter must be an alphabet or underscore (_).
- Identifiers are case sensitive.
- Reserved keywords can not be used as an identifier's name.

Constants

Constants are like a variable, except that their value never changes during execution once defined.

There are two other different ways to define constants in C++. These are:

- By using const keyword
- By using #define preprocessor

Declaration of a constant :

```
const [data_type] [constant_name]=[value];
```

Variable

A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer.

Syntax to declare a variable

```
[data_type] [variable_name];
```

Example

```
#include <iostream.h>

int main() {

   int a,b;// a and b are integer variable

   cout<<" Enter first number :";

   cin>>a;

   cout<<" Enter the second number:";

   cin>>b;

   int sum;

   sum=a+b;

   cout<<" Sum is : "<<sum <<"\n";

   return 0;

}
```

Operator

C++ operator is a symbol that is used to perform mathematical or logical manipulations.

- Arithmetic Operators
- Increment and Decrement Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Increment and Decrement Operators

Operator	Description
++	Increment
--	Decrement

Relational Operators

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less
>=	Greater than or equal to

<=	Less than or equal to

Logical Operators

Operator	Description
&&	And operator. Performs logical conjunction of two expressions.(if both expressions evaluate to True, result is True. If either expression evaluates to False, the result is False)
||	Or operator. Performs a logical disjunction on two expressions.(if either or both expressions evaluate to True, the result is True)
!	Not operator. Performs logical negation on an expression.

Bitwise Operators

Operator	Description
<<	Binary Left Shift Operator
!=	Is not equal to
>>	Binary Right Shift Operator
~	Binary One's Complement Operator
&	Binary AND Operator
^	Binary XOR Operator
|	Binary OR Operator

Assignment Operators

Operator	Description
=	Assign
+=	Increments, then assign
-=	Decrements, then assign
*=	Multiplies, then assign
/=	Divides, then assign
%=	Modulus, then assigns
<<=	Left shift and assigns
>>=	Right shift and assigns
&=	Bitwise AND assigns
^=	Bitwise exclusive OR and assigns
|=	Bitwise inclusive OR and assigns

Misc Operators

Operator	Description
,	Comma operator
sizeOf()	Returns the size of a memory location.

&	Returns the address of a memory location.
*	Pointer to a variable.
? :	*Conditional Expression*

The preprocessors are the directives, which give instructions to the compiler to preprocess the information before actual compilation starts.

How to declare ?

All preprocessor directives begin with #, and only white-space characters may appear before a preprocessor directive on a line. Preprocessor directives are not C++ statements, so they do not end in a semicolon (;).

Example:

```
#include<iostream>

using namespace std;

int main()

{
    // prints hello world
    cout<<"Hello World";

    return 0;
}
```

Here, #include preprocessor directive and it makes header file like iostream available for us i.e. includes header file in our program.

There are number of preprocessor directives supported by C++ like #define, #if, #else, #line, etc.

THERE ARE 4 MAIN TYPES OF PREPROCESSOR DIRECTIVES.

1. **Macros**
2. **File Inclusion**
3. **Conditional compilation**
4. **Other directives**

- **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.

- **File Inclusion**

This type of preprocessor directive tells the compiler to include a file in the source code program. There are two types of files which can be included by the user in the program

1. Header file and Standard files.
2. User defined files.

- **Conditional compilation**

Conditional Compilation directives are type of directives which helps to compile a specific portion of the program or to skip compilation of some specific part of the program based on some conditions. This can be done with the help of two preprocessing commands '**ifdef**' and '**endif**'.

- **Other directives**

Apart from the above directives there are two more directives which are not commonly used. These are:

1. **#undef directive**: The #undef directive is used to undefine an existing macro.

   #undef LIMIT

   Using this statement will undefine the existing macro LIMIT. After this statement every "#ifdef LIMIT" statement will evaluate to false

2. **#pragma directive**: This directive is a special purpose directive and is used to turn on or off some features. This type of directives are compiler-specific, i.e., they vary from compiler to compiler.

   Here, are few examples of #pragma directives.
   a. #pragma startup
   b. #pragma exit

3. **#pragma warn directive**: This directive is used to hide the warning message which are displayed during compilation.
   We can hide the warnings as shown below:

   - **#pragma warn -rvl**:
     This directive hides those warning which are raised when a function which is supposed to return a value does not returns a value.

   - **#pragma warn -par**:
     This directive hides those warning which are raised when a function does not uses the parameters passed to it.

   - **#pragma warn -rch**:
     This directive hides those warning which are raised when a code is unreachable. For example: any code written after the *return* statement in a function is unreachable.