Basics of C++ Programming

Introduction: The C++ programming language was developed at AT&T Bell Laboratories in the early 1980s by Bjarne Stroustrup.

C++ Character Set:

Character set is a set of valid characters that a language can recognize. A character represents any letter, digit or any other sign. The C++ has the following character set:

Letters: A-Z, a-z

Digits: 0-9

Special Symbols: space, + - * / ^ \ ( ) [ ] { } = != < > . ' "
$, ; : % ! & ? _(underscore) # <= >= @

Whitespaces: Blank space, Horizontal tab (→), Carriage return (↵)...

Other characters: C++ can process any of the 256 ASCII characters as data or as literals.

Tokens (Lexical Units):

Token: The smallest individual unit in a program is known as a Token or a lexical unit. C++ has the following tokens:

Keywords
Identifiers
Literals
Punctuators
Operators:

Keyword: keyword is a word having special meaning reserved by programming language.

C++ contains the following keywords:

asm	char	delete	extern	if	operator
auto	const	do	float	inline	private
break	class	double	for	int	protected
case	continue	else	friend	long	public
catch	default	enum	goto	new	register

return	static	this	union	while	false
short	struct	throw	unsigned	using	
signed	switch	try	virtual	namespace	
sizeof	template	typedef	void	true	

Identifiers: Identifiers are fundamental building blocks of a program and are used as the general terminology for the names given to different parts of the program.

Eg: variables, objects, classes, functions, arrays etc.

C++ is case sensitive as it treats upper and lowercase characters differently.

The following are some valid identifiers:

Myname   _file
(Underscore)

The following are some invalid identifiers.

add-sub	contains special character -
9cl	Starting with a digit
break	reserved keyword
My.name	contains special character. dot(.)

Literals Literals (often referred ~~as~~ to as constants) are data items that never change their value during a program run.

Eg: true, false, +97, -16, c, A, 2.0, 1.7E, "etc"

Punctuators:

The following characters are used as punctuators (also known as separators

[ ] ( ) { } , ; : * ... = #

[ ] indicates single and multidimensional array subscripts

( ) indicate function calls, function parameters and for expressions.

{ } indicate start and end of a compound statement.

Comma ,

It is used as a separator in a function argument list.

Semicolon ;

It is used as a statement terminator

Colon :

It indicates a labeled statement.

Asterisk *

It is used for pointer declaration.

Ellipsis ...

These are used in the formal argument lists of function prototypes to indicate a variable number of argument.

Equal to sign =

It is used for variable initialisation and as assignment operator.

pound sign (#)

It is used for preprocessor directive.

Operators: Operators are tokens that trigger some computation when applied to variable and other objects in an expression.

Operators are of two types.

1. Unary operators
2. Binary operators.

1. Unary operators: These are the operators that require one operand to operate upon. The following are some unary operators.

&	Address operator
*	Indirection operator
+	Unary plus
-	Unary minus
~	Bitwise Complement
++	increment operator
--	decrement operator
!	Logical negation

2. Binary operators: Binary operators are those operators that require two operands to operate upon.

Arithmetic operators. + (Addition), - (Subtraction) * (multiplication) / (Division) % (Remainder/Modulus)

· Shift operators: << shift left >> shift right

Bitwise operators: Bitwise AND, Bitwise exclusive OR (XOR) Bitwise OR

Logical operators: && Logical AND || Logical OR

Assignment operators: (=) Assignment (/=) Assignment quotient
(+=) Assign sum (<<=) Assign left shift (&=) Assign bitwise AND
(*=) Assign product (%=) Assign remainder (-=) Assign difference
(>>=) Assign right shift (^=) Assign bitwise XOR

Relational operators
<, >, <=, >=, ==, !=

Component selection operators
. Direct component selector
-> Indirect component selector

Class member operators
:: scope access/resolution
.* Deference pointer to class member
->* Deference pointer to class member

Conditional operators
?
:

The order of precedence for operators is as follows

```
() [] ->                                   &&
! ~ + - ++ -- & * (typeset)                ||
* / %                                      ?:
+ -                                        ~~etc~~
<< >>
<<= >>=
== !=
&
^
|
```