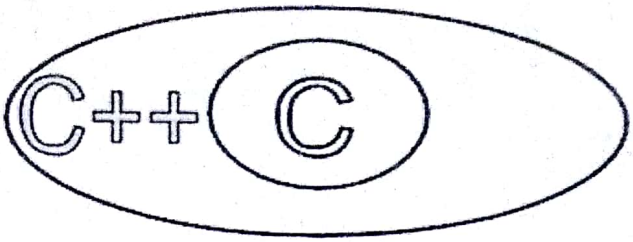

C++ was developed by Bjarne Stroustrup in the begining of 1979 at Bell Labs as a free-form, multi-paradigm, compiler based, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises both high-level and low-level language features. C++ was originally named as C with Classes, adding object oriented features, such as classes, and other enhancements to another already existing C programming language. The language was renamed C++ in 1983, as it has an increment operator ++.

C++ is one of the most popular programming languages and is implemented on wide variety of hardware and operating system platforms. As an efficient compiler to native code, it is widely used for developing systems software, application software, device drivers, embedded software, high-performance client-server applications and entertainment software such as video games.

C++ has greatly influenced many other popular programming languages such as C# and Java.

Smallest possible C++ program that will compile with no errors:

On Turbo C++ compiler	On some other C++ compilers

`void main()`
`{`
 	
`}`
	`int main()`
	`{`
	`  return 0;`
	`}`

Let us first start with a program, which will display some output on the screen:

`#include <iostream.h>`
`void main()`
`{`
`  cout<<"This is my first program!"`
`}`
	`#include <iostream>`
	`int main()`
	`{`
	`  cout<<"This is my first program!";`
	`  return 0;`
	`}`

OUTPUT	OUTPUT
`This is my first program!`	`This is my first program!`

In the above program, we have included a header file **iostream.h** to use the output command **cout** using **#include**. Here # is a pre-processor directive, which instructs compiler to add the commands before starting the main compilation process.

Some of the common C++ compilers supported in Windows as follows:
Turbo C++ for DOS, Dev C++, Visual C++, Open Watcom, CodeWarrior, etc.

C++ character set is a set of valid characters that a language can recognise and understand.

Letters	A-Z, a-z
Digits	0-9
Special Characters	Space + - * / ^ \ () [] {} = != <> ' " $, ; : % ! & ? _ # <= >= @
Formatting characters	backspace, horizontal tab, vertical tab, form feed, and carriage return

In a C++ program, Tokens are the smallest unit, which are logically grouped characters. Keywords, Identifiers, Literals, Operators and punctuators are basic C++ tokens.

Keyword: A keyword is a reserved word that is pre-defined in the C++ compiler for a particular purpose. An identifier can not have the same name as keyword. Some of the C++ keywords are as follows:

break	delete	if	public	switch
case	do	int	return	this
char	double	long	short	typedef
class	else	new	signed	unsigned
const	float	private	sizeof	void
default	for	protected	struct	while

Identifiers: It is a symbolic name given by the programmer to any data item or unit of the program. Rules to name a C++ identifier are as follows:

- Can consist of Alphabets, Digits and/or UnderScore (_)
- Can start with an Alphabet/UnderScore (_) but not with digit
- Can not have a space in it
- Can not be a keyword
- C++ is case sensitive and so upper case and lower case letters are considered different from each other. i.e., AMOUNT and Amount are two different identifiers.

Literals (constants) are data items, whose value will never change during the execution of th program. The following types of literals are available in C++.

• Integer-Constants	456	-567	0
• Character-constants	'A'	'm'	'*'
• Floating-constants	45.987	-543.12	0.0
• Strings-constants	"DELHI"	"RAM KUMAR"	"I have 2 pens"

Punctuators: The following is the list of characters used as punctuators in C++.

Brackets [ ]	Opening and closing brackets indicate single and multi dimensional array subscript.
Parentheses ( )	Opening and closing brackets indicate function calls for enclosing parameters.
Braces { }	Opening and closing braces indicate the start and end of a compound statement.
Comma ,	Used as a separator in a variable declaration or function argument list.
Semicolon ;	Used as a statement terminator.
Colon :	Indicates a labeled statement or conditional operator symbol.
Asterisk *	Used in pointer declaration or as multiplication operator.
Equal to sign =	Used as an assignment operator.
Pound sign #	Used as pre-processor directive.

Operators: Operators are special symbols used in C++ for specific purposes. C++ has six types of operators. Arithmetic, Relational, Logical, Assignment, Conditional, Comma and Unary operators.

Arithmatic Operators are as follows:

+	Addition	-	Subtraction	*	Multiplication
/	Division	%	Remainder		

Other operators will be discussed later.

A sample program using arithmatic operators is as follows:

```
#include <iostream.h>
void main()
{
  cout<<45+90<<endl;
  cout<<4+5*2<<endl;
  cout<<4*5+9*10<<endl;
  cout<<15%4<<endl;
  cout<<(4+5)*9-10<<" and ";
  cout<<4/2*3<<endl;
}
```

OUPUT

```
135
14
110
3
71 and 6
```

Here, endl is used in cout for denoting end of line in the outout.

Data Type: While discussing about constants, you have already seen the categorization of various kind of values that can be used in a C++ program such as character, integer and float (real numbers). C++ has four basic data types to represent constants and variables - **char** represents character type, **int** represent integer type, **float** represent real numbers and **double** represent large size real numbers.

Variable: It is an identifier (a symbolic representation) referring to a memory location, which can take any value of a particular data type. It is essential to declare a variable belonging to a particular data type before using it in the program.

Syntax:

```
<Data Type> <Variable1>[,<Variable2>,<Variable3>,...];
```

Examples of declaration of variables are as follows:

`int Age,Score;`	`float SalePrice;`	`double Budget;`	`char Grade,Gender;`

Initialization: A variable can also be assigned some initial value at the time of declaration.

Syntax:

```
<Data Type> <Variable1>=<Value1>[,<Variable2>=<Value2>,...];
```

For example

`int Score=0;`	`float Total=1000.50;`	`char Gender='M';`

Assignment: In C++, we can assign a variable with value of another variable, constant and expression.

Syntax:

```
<Variable>=<Variable/Expression/Constant>;
```

For example

`Score=45;`	`Total=Sum+Amount;`	`MyGrade=Grade;`

Access modifier: In C++, we can declare an identifier to assign a constant value, i.e. a symbolic representation for a constant, whose value will remain same through out the program using a keyword **const.**

Syntax:

```
const <Data Type> <Constant Identifier>=<Constant Value>;
```

For example:

```
const int Age=60;
void main()
{
 int MyAge=5;
 cout<<"60 Years Later"
     <<MyAge+Age;
}
```

```
const float PIE=3.1416;
void main()
{
  float Radius=5,Area;
  Area=PIE*Radius*Radius;
  cout<<"Area:"<<Area<<endl;
}
```

```
const char Male='M';
void main()
{
  char Gender;
  Gender=Male;
  cout<<Gender;
}
```

Type modifier: Type modifiers in C++ are used to modify the range or/and size of a particular basic data type. The modifiers signed, unsigned, long, and short can be applied to int type. In addition, signed and unsigned can be applied to char, and long can be applied to double.

The modifiers signed and unsigned can also be used as prefix to long or short modifiers. For example unsigned long int.

Data Type	No. of Bytes	Minimum Value	Maximum Value
char	1	0	255
int	2	-32768	+32767
signed char	1	-128	+127
unsigned char	1	0	255
unsigned int	2	0	65535

signed char ch; (value decrease) (range)

long only add 2 bytes of memory.
int → 2 bytes
long int → 4 bytes
double → 8 bytes
long double → 10 bytes

(*) long is only used for int and double

signed int	2	-32768	+32767
short int	2	-32768	+32767
long int	4	-2147483648	+2147483647
unsigned long int	4	0	+4294967295
signed long int	4	-2147483648	+2147483647

! The above table contains bytes and range as per 16 bit compiler. It varies for 32 bit and 64 bit compilers.

long int A ; ≡ long A ;

Now, let us see a complete program with INPUT-PROCESS-OUTPUT

SAMPLE PROGRAM	OUTPUT

`#include <iostream.h>`
`void main()`
`{`
`  float Principle,Rate,Time,SimpleInterest;`
`  cout<<"Principle:";cin>>Principle;`
`  cout<<"Rate :";cin>>Principle;`
`  cout<<"Time :";cin>>Time;`
`  SimpleInterest=Principle*Rate*Time/100;`
`  cout<<"Simple Interest:"<<SimpleInterest<<endl;`
`}`
	`Principle:5000`
	`Rate :10`
	`Time :5`
	`Simple Interest:2500`

In the above program, we have used cin with >> operator to take the input from the user. << operator is known as **extraction operator.** cin is also part of the **iostream** header file. cin can also be used to allow user to enter multiple values for multiple variable. For example: **cin>>Amount>>Discount;** In such case, each variable will be separated by >> operator. You can recall, we used << operator with cout, which is known as **insertion** operator.

Source Code/File: The program written by you using C++ commands is known as Source Code. The source code (with a defualt file extension .cpp) needs to be compiled to **executable** code (with extension .exe), so that it can be executed to give desired result.

Syntax error: If a program does not follow the correct syntax of the language, compilation process terminats and results in **syntax error.** Examples of **Syntax Errors** are as follows:

`cout>>Amount>>Price;`	Incorrect use of >> operator
`X+Y=Z;`	

Logical Error: These errors occur due to wrong use of formula or expression by the programmer. Due to this, the program produces wrong results while execution. Example of **Logical Error**

If the following statement is written by the programmer to calculate Average marks of three subjects, it is going to be a Logical error.	If the following statement is written by the programmer to calculate Area of rectangle, it is going to be a Logical error.
`Avg=English+Maths+Science/3;`	`Area=Length+Bredth;`

Run-Time/Execution-Time Error: These errors are encountered during execution of the program due to unexpected input or output or calculation. Example of **Run-Time/Execution-Time Error**

`int A,B,C;`
`cin>>A>>B;`
`C=A/B;`
`cout<<C<<endl;`
	The program shown here will result in run-time error (division by zero), if user enters 0 for the variable B.

C++ shortcuts:

`+=`	`A+=B;`	`A=A+B;`
`-=`	`A-=B;`	`A=A-B;`
`*=`	`A*=B;`	`A=A*B;`
`/=`	`A/=B;`	`A=A/B;`
`%=`	`A%=B;`	`A=A%B;`

Unary operators: The operators, which work with only one operand are known as a unary operators. Examples of unary operators are as follows:

Minus Operator	Increment Operator	Decrement Operator
-	++	--

`int A=100,B;`
`B=-A;`
`cout<<B<<endl;`
	`int A=100;`
	`A++;`
	`cout<<A<<endl;`
	`++A;`
	`cout<<A<<endl;`
		`int A=100;`
		`A--;`
		`cout<<A<<endl;`
		`--A;`
		`cout<<A<<endl;`

`OUTPUT`
`-100`
	`OUTPUT`
	`101`
	`102`
		`OUTPUT`
		`99`
		`98`

Pre-Increment/Post-Increment and Pre-Decrement/Post-Decrement operators: When the increment/decrement operator is placed before the operand (variable), it is known as pre-increment operator. And, when the increment/decrement operator is placed after the operand (variable), it is known as post-increment operator. In the above examples, you see no impact of Pre/Post placement of Increment/Decrement operators as these are used in independant statements. Pre/Post placement of Increment/Decrement impacts when used in a part of a statement/an expression. Examples of pre/post increment operators are as follows:

`int A=100,B=200;`
`A+=B++;`
`cout<<A<<"#"<<B<<endl;`
`A+=++B;`
`cout<<A<<"@"<<B<<endl;`
	`int A=100,B=200;`
	`cout<<A+B<<"#"<<++A<<"#"<<B++<<endl;`
	`cout<<A++<<"#"<<++B<<"#";`
	`cout<<A+B<<endl;`

`OUTPUT`
`300#201`
`502#202`
	`OUTPUT`
	`302#101#200`
	`101#202#304`