Introduction

C++, in Computer Science, an object-oriented version of the C programming language, developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories and adopted by a number of vendors, including Apple Computer, Sun Microsystems, Borland International and Microsoft Corporation.

Data Types

In C++ there are **five (5)** Fundamental (Built-in or Primitive or Basic) data types:

1. **`void`** represents no type (type less) and usually used as a return value of a function. Data type **`void`** is the odd one out since we cannot define a variable of the type **`void`**.
2. **`char`** represents any single character from keyboard enclosed within a single quotes ('). Every computer has a character set. PC has ASCII character set. ASCII character set has 256 characters. Data type char represent any character from ASCII character set.
3. **`int`** represents zero, positive and negative integer values (whole numbers).
4. **`float`** represents zero, positive and negative floating point values (real numbers).
5. **`double`** represents zero, positive and negative floating point values (real numbers). Data type **`double`** is similar to **`float`** but with better precision.
6. String: represents sequence characters enclosed within a pair of double quotes ("). **String is not a fundamental type**. String is a derived data type.

Constants

A value which is hard coded into a program, which remains unchanged through out the program. Constants are of **five (5)** types:

1. **`char`** constant	Character constant
2. **`int`** constant	Integer constant
3. **`float`** constant	Single precision floating point constant
4. **`double`** constant	Double precision floating point constant
5. String constant	sequence of characters enclosed within a pair of double quotes (").

Note: C++ does not support constant of the type **`void`**.

Examples of C++ Constants are given below:

Data Type	Constants
char	'A','B','C',…,'X','Y','Z' - uppercase constant 'a','b','c',…,'x','y','z' - lowercase constant '0','1',…,'8','9' - digits '!','@','%','^','&','*','+',…,'?' - special characters
int	1, 2, 3, …, 2147483647 - positive integer constant 0 - zero -1, -2, -3, …, -2147483648 - negative integer constant
float **double**	0.5, 1.5, 20.75 - positive floating point constant 0.0 - zero -0.3, -2.8, 15.68 - negative floating point constant
string	"Amit", "Pizza", "India", "Apple", "G", "9", "$", "e" "GH-14/12", "23981535", "20/10/2005", "***", "6.0", ""

Variables

A variable is name given to a memory location to store value in the computer's main storage (RAM). It is a name used in the program that represents data (value). The value assigned to the variable name may change (vary) as the program is executed. The program can always access the current value of the variable by referring to its name. In C++ variables are to be created before they can be used. To create a variable we need to give a name to a variable.

Rules for naming a C++ variable (identifier)

1. Variable name should start with an alphabet (letter) or an underscore.
2. Variable name may contain more than one character. Second characters onwards we may use only alphabets or digit or underscore.
3. No special characters are allowed in a variable name except underscore.
4. A variable name in C++ is case sensitive. Uppercase and lowercase letters are distinct.

   `Sum`, `sum`, `SUM` and `sUm` are treated as four different variable names in C++.

5. A variable name cannot be a keyword.

   `void`, **`char`**, **`int`**, **`float`**, **`double`**, **`if`** and **`else`** are incorrect variable names because they are keywords.

6. In Borland C++ only first 55 characters in a variable name are significant.

   Examples of correct variable names are given below:
   `marks, m1, m2, Father_Name, Max_Score, sub1code, ans, Roll, INT, Char, _Val, _Input_Screen, CompScMarks`

   Generally a C++ variable name does not start with an underscore (_). List of incorrect variable names are given below:

In correct Variable Name	Reasons
`1m`, `2ndst`, `#No`, `%Att`	Variable name starts with either digit or special character
`Stu-name`, `name$`, `marks 1`, `val%`, `GH-8/64`	Variable names contain either digit or special character

Creating variable

A variable is a name given to a memory location to store a value and it represents a value in a program. The value assigned to the variable name may change during execution of program. The program can always access the current value of the variable by referring to its name.

```
Rule: DataType VariableName;
      DataType VariableName1, VariableName2, VariableName3, … ;
```

Usage

```
char sex;
char ans, choice, section;
char name[30];
char subject[20], country[25];

int roll;
int flatno, ecode, mobile, phone;

float average;
float area, length, marks;

double temperature;
double radius, price, rate;
```

Creating a variable is a statement in C++ and every C++ statement is terminated by a semi-colon (;). String is not a fundamental data type but still examples are given how to create string variables. An array of character is used to create a string variable. An example is given below:

```
char name[30];
```

More detailed discussion about array and strings will be done later.

Memory Allocation

Every variable in C++ is allocated fixed amount of memory. C++ data types, memory allocation and range of values are given below:

Date Type	Storage (Memory Allocation)	Range of values		
1. **char**	1 byte or 8 bits	-128	to	127
2. **int**	4 bytes or 32 bits	-2147483648	to	2147483647
3. **float**	4 bytes or 32 bits	$3.4*10^{-38}$	to	$3.4*10^{38}$
4. **double**	8 bytes or 64 bits	$1.7*10^{-308}$	to	$1.7*10^{308}$

Assignment Operator

Value is assigned to a variable by using assignment operator. Using assignment operator, value is stored in a variable when writing a program. Using assignment operator, value is copied to a variable.

```
Rule:  VariableName = Value;
       DataType VariableName = Value;
```

Usage of assignment operator

```
char ch;
int roll;
double rate;
ch='A';
sum=13;
rate=154.25;
```

Usage of assignment operator

```
char ch='A';
int roll=13;
double rate=154.25;
```

Console Output (cout)

Using cout and **output operator** (<<) value can be displayed on the screen (console). A list of data items can be displayed with single cout, each data separated by output operator (<<).

```
Rule:  cout<<Value;
       cout<<Value1<<Value1<<Value3…;
       cout<<Value<<endl;
```

Usage of cout

```
cout<<"VINAY AHUJA";
cout<<11;
cout<<'A';
cout<<78.5;
```

Produces output like

```
Vinay Ahuja11A78.5
```

Without endl, next output is displayed immediately after previous output. As a result all four (4) data items are displayed next to each other without any space.

Usage of cout

```
cout<<"VINAY AHUJA"<<11<<'A'<<78.5;
```

Produces output like

```
VINAY AHUJA11A78.5
```

Usage of `cout`

```
cout<<"VINAY AHUJA"<<endl;
cout<<11<<endl;
cout<<'A'<<endl;
cout<<78.5<<endl;
```

Produces output like

```
VINAY AHUJA
11
A
78.5
```

With `endl`, next output is displayed in the beginning of the next line. As a result all four(4) data items are displayed on four separate lines.

Usage of `cout`

```
char name[20]="VINAY AHUJA";
int cla=11;
char sec='A';
double marks=78.5;
cout<<"Name   ="<<name<<endl;
cout<<"Class  ="<<cla<<endl;
cout<<"Section="<<sec<<endl;
cout<<"Marks  ="<<marks<<endl;
```

Produces output like

```
Name   =VINAY AHUJA
Class  =11
Section=A
Marks  =78.5
```

Displaying many values by using single `cout` and separating the values by output operator (<<) is known as **cascading of output operator**. An example is given below:

```
cout<<"VINAY AHUJA"<<11<<'A'<<78.5<<endl;
```

Console Input (`cin`)

Using `cin`, value can be inputted in a variable when a program is getting **executed (running)**. `cin` causes a program to stop and wait for user to input value through a keyboard (console). It will then store the value inputted in a variable. A variable is to be created (defined) and then value can be inputted by using `cin`. List of value can be inputted using `cin`, separating the variable names by input operator (>>).

```
Rule: cin>>VariableName;
      cin>>VariableName1>>VariableName2>>VariableNname3 …;
```

Usage of `cin`

```
char name[20], sec;
int cla;
double marks;
```

```
cin>>name;
cin>>cla;
cin>>sec;
cin>>marks;
```

Produces a screen like

```
VINAY↵
11↵
A↵
78.5↵
```

After every input Enter key (↵) is pressed. When inputting a string (`Vinay`) double quotes (") are not required. When inputting a character (`A`) single quote quotes (') are to be avoided.

Usage of `cin`

```
char name[20];
int cla;
char sec;
double marks;
cin>>name>>cla>>sec>>marks;
```

Produces a screen like

```
VINAY□11□A□78.5↵
```

Every input is separated by Space (□) but final key stroke is Enter (↵).

Or,

Produces a screen like

```
VINAY→11→A→78.5↵
```

Every input is separated by Tab (→) but final key stroke is Enter (↵).

Inputting many values by using single `cin` and separating the variable names by input operator (>>) is known as **cascading of input operator**. An example is given below:

```
char name[20];
int cla;
char sec;
double marks;
cin>>name>>cla>>sec>>marks;
```

To make an input more user friendly, it is better to display a prompt or a message before an input so that the user knows exactly what kind of input is required for the program.

Usage of prompt or message with `cin`

```
int cla;
char name[20], sec;
double marks;
cout<<"Input Name   ? "; cin>>name;
cout<<"Input Class  ? "; cin>>cla;
cout<<"Input Section? "; cin>>sec;
cout<<"Input Marks  ? "; cin>>marks;
```

After execution of above program segment produces a screen like this

```
Input Name   ? VINAY
Input Class  ? 11
Input Section? A
Input Marks  ? 78.5
```

Structure of a C++ program

A complete C++ program consists of header files and at least one function (main() function). The most important function in C++ is the main() function. A complete C++ program may contain other functions as well, but they are invoked from the main() function only. An example is given below:

```
#include<iostream.h>
#include<conio.h>
void main()
{
   cout<<"This is my first program using C++";
   getch();
}
```

Running of the program will produce output screen like

```
This is my first program using C++
```

Note: Output remains on the screen till you press any key because getch() function waits for a user to strike any key.

Header files: The header file iostream.h is required for cout and output operator (<<). To use the function getch() we need the header file conio.h. C++ compiler will obtain necessary information about cout, << and getch() from the header files.

Function: A C++ function has two main components – header and block (body).
Function Header – **void** main()
Function block

```
{
   cout<<"This is my first program using C++";
   getch();
}
```

A block starts with curly bracket ({) and ends with curly bracket (}). Every C++ function contains C++ statements. Every C++ statement is separated by a semi-colon (;).

Some important keyboard shortcuts:

a) To **compile** a program press **ALT+F9** (Click Project from Menu Bar and then click Compile). Compiler will convert a program written in high level language (source code – CPP file) into an intermediate machine language code (Object Code – OBJ file). Compiler also checks for syntax errors. Object code will be successfully generated provided the Source Code does not contain any syntax error(s).
b) To **make** (**compile** and **link**) a program press **F9** (Click Project from Menu Bar and then click Make). A linker will add Run-Time Library to the Object Code to obtain Executable Machine

Language Code (Executable Code – EXE file). A computer or the CPU (processor) of the computer can only execute Machine Language Code (EXE file). Run-Time Library is collection sub-routines needed to run a program.

c) To **run** (**compile**, **link** and **execute**) a program press **CTRL+F9** (Click Debug from Menu Bar and then click Run). Machine Language Executable file is loaded in the computer's main storage from the computer's secondary storage and the program is executed. When the program is getting executed, a DOS Window pops up on the Desktop. DOS Window disappears after the execution of the program.

```
#include<iostream.h>
#include<conio.h>
void main()
{
   char name[20], sec;
   int cla;
   double marks;
   cout<<"Input Name   ? "; cin>>name;
   cout<<"Input Class  ? "; cin>>cla;
   cout<<"Input Section? "; cin>>sec;
   cout<<"Input Marks  ? "; cin>>marks;
   cout<<"Name   ="<<name<<endl;
   cout<<"Class  ="<<cla<<endl;
   cout<<"Section="<<sec<<endl;
   cout<<"Marks  ="<<marks<<endl;
   getch();
}
```

Running of the program produce screen like

```
Input Name   ? VINAY↵
Input Class  ? 11↵
Input Section? A↵
Input Marks  ? 78.5↵
Name   =Vinay
Class  =11
Section=A
Marks  =78.5
```

Running of the program produce screen like

```
Input Name   ? VINAY AHUJA↵
Input Class  ? Input Section? Input Marks  ? Name   =VINAY
Class  =1
Section=
Marks  =1.84513e-307
```

Inputting string with a space creates run-time error. C++ program treats space as separator. Remaining three inputs are ignored. String `"VINAY"` get stored in the variable `name`. Garbage values get stored in the variable `cla`, `sec` and `marks`.

Arithmetic Operators

C++ supports all the four arithmetic operators like plus (addition +), minus (subtraction -), multiplication (product *) and division (divide /). In addition to this it supports remainder operator

(%). Remainder operator (%) can only be used with integer type (**int** type). Brackets or parenthesis () are also supported by C++. Operators *, / and % are given more precedence compared to operatots + and -. However operators *, / and % are given same precedence. Similarly operators + and – are given same precedence.

Operator	Meaning	Usage	Result
+	Addition	`10 + 20`	`30`
		`72.75 + 57.65`	`130.4`
–	Subtraction	`30 - 15`	`15`
		`23 - 57`	`-34`
		`42.5 - 19.25`	`23.25`
		`176.5 - 225.25`	`-48.75`
*	Multiplication	`12 * 18`	`216`
		`7.5 * 2.5`	`18.25`
/	Division	`40 / 5`	`8`
		`14 / 4`	`3`
		`4 / 10`	`0`
		`12.5 / 2.5`	`5.0`
		`13.5 / 20`	`0.675`
		`10.2 / 0 or 7 / 0`	`Run-Time Error`
%	Remainder	`15 % 4`	`3`
		`4 % 10`	`4`
		`10 % 2.5`	`Syntax Error`
		`5 % 0`	`Run-Time Error`
()	Parenthesis	`(2 + 3) * (6 - 3)`	`15`
		`28 / (16.5 - 9.5)`	`4`

Operator	Precedence
()	Expressions within parentheses are evaluated first
`* / %`	Multiplication, division and remainder are evaluated next
+ –	Addition and subtraction are evaluated last

Numeric Expression

A C++ expression involving Arithmetic Operators is called numeric expression. Any expression in C++ consists of operators and operands. Examples of C++ numeric expressions are given below:

Expression	Operator	Operands
`10 + 20`	`+`	`10` and `20`
`25 - 16`	`-`	`25` and `16`
`35 / 4.25`	`/`	`35` and `4.25`
`20 * 1.25`	`*`	`20.5` and `1.25`
`25 % 7`	`%`	`25` and `7`
`35 * 2`	`*`	`35` and `2`

Pure Expression: An expression where all the operands belong to same data type.

Rule: **int** operator **int** = **int**
float operator **float** = **float**
double operator **double** = **double**

Examples of pure expressions:

Integer Type	Floating Point Type
`10 + 20`	`2.5 + 3.8`
`20 - 5`	`9.8 - 3.5`
`17 * 6`	`11.25 * 2.5`
`35 / 7`	`5.7 / 1.9`
`34 % 5`	`10.8 / 3.2`

Mixed Expression: An expression where the operands belong to different data types.

Rule: **int** operator **char** = **int**
char operator **int** = **int**
int operator **float** = **float**
float operator **int** = **float**
int operator **double** = **double**
double operator **int** = **double**

Examples of mixed expressions:

`32 + 'A' = 97` since ASCII code of `'A'` is 65
`'t' - 32 = 84` since ASCII code of `'t'` is 116
`20.0 / 8 = 2.5`
`20 + 2.5 = 22.5`

```
#include<iostream.h>
void main()
{
   int a, b;
   cout<<"Input two integers? "; cin>>a>>b;
   int su=a+b, pr=a*b, di=a-b, qu=a/b, re=a%b;
   cout<<"Sum       ="<<su<<endl;
   cout<<"Product   ="<<pr<<endl;
   cout<<"Difference="<<di<<endl;
   cout<<"Quotient  ="<<qu<<endl;
   cout<<"Remainder ="<<re<<endl;
}
```

Execution of the program produces output screen

```
Input two integers? 15 5↵
Sum       =20
Product   =75
Difference=10
Quotient  =3
Remainder =0
```

Value `15` get stored in ‘`a`’ and value `5` get stored in ‘`b`’. Since ‘`a`’ is perfectly divisible by ‘`b`’, therefore quotient is `3` and remainder is `0`.

Execution of the program produces output screen

```
Input two integers? 20 6↵
Sum       =26
Product   =120
Difference=14
Quotient  =3
Remainder =2
```

Value `20` get stored in ‘`a`’ and value `6` get stored in ‘`b`’. Since ‘`a`’ is not perfectly divisible by ‘`b`’, therefore quotient is `3` (integer part of the quotient) and remainder is `2`.

Execution of the program produces output screen

```
Input two integers? 10.5 2↵
Sum       =4243922
Product   =42439120
Difference=-4243902
Quotient  =0
Remainder =10
```

Value 10 (integer part of 10.5) get stored in 'a' and inputted value 2 is ignored. As a result 'b' stores garbage value (4243922). Therefore output is also garbage values.

Execution of the program produces output screen

```
Input two integers? 10 2.5↵
Sum       =12
Product   =20
Difference=8
Quotient  =5
Remainder =0
```

Value 10 get stored in 'a' and inputted value 2 (integer part of 2.5) get stored in 'b'. Since there is no other input after 2.5, floating input for an integer variable does not create any problem.

```
#include<iostream.h>
void main()
{
   double a, b;
   cout<<"Input two values? "; cin>>a>>b
   double su=a+b;
   double pr=a*b;
   double di=a-b;
   double qu=a/b;
   cout<<"Sum       ="<<su<<endl;
   cout<<"Product   ="<<pr<<endl;
   cout<<"Difference="<<di<<endl;
   cout<<"Quotient  ="<<qu<<endl;
}
```

Execution of the program produces output screen

```
Input two values? 22.5 2.5↵
Sum       =25
Product   =56.25
Difference=20
Quotient  =9
```

Execution of the program produces output screen

```
Input two values? 15 4↵
Sum       =19
Product   =60
Difference=11
Quotient  =3.75
```

Execution of the program produces output screen

```
Input two values? 12.3 5↵
Sum       =17.3
Product   =61.5
Difference=7.3
Quotient  =2.46
```

When inputting floating value in a floating point variable, one can input floating point value and integer value as well. But integer input will be converted into a floating point value and then it will be stored in the floating point variable.

- When the program is executed for the first time, only floating values are inputted.
- In the second execution only integer values are inputted.
- In the third execution a floating point value and an integer value is inputted.

```
#include<iostream.h>
void main()
{
   double a, b, c;
   cout<<"Input 3 value? ";
   cin>>a>>b>>c;
   double res1=a+b*c;
   double res2=a+b-c;
   double res3=(a+b)/c;
   cout<<"Res1="<<res1<<endl;
   cout<<"Res2="<<res2<<endl;
   cout<<"Res3="<<res3<<endl;
}
```

double res1=a+b*c;
First b*c is evaluated and result is added to a since * has higher than +.

double res2=a+b-c;
First a+b is evaluated and c subtracted from the result. + and – have same precedence; therefore + is operation is carried out first.

double res3=(a+b)/c;
First a+b is evaluated and the result is divided by c. + has lesser than / but parenthesized expression is evaluated first.

ALGEBRAIC EXPRESSION

An expression involving arithmetic operators and arithmetic function is called Algebraic Expression. Examples of algebraic expression and equivalent C++ expression are given below:

Algebraic Expression	C++ Expression
$a+b$	a + b
$a-b$	a - b
ab	a * b
$\frac{a}{b}$	a / b
a^b	pow(a,b)
$a^2+2ab+b^2$	pow(a,2) + 2*a*b + pow(b,2) a*a + 2*a*b + b*b
$\sqrt{a}$	sqrt(a), pow(a, 0.5), pow(a, 1.0/2)

$\sqrt{a^2+b^2}$	sqrt(pow(a,2) + pow(b,2)), sqrt(a*a + b*b)
	pow(pow(a,2) + pow(b,2), 0.5)

a^4	pow(a, 4),a*a*a*a

$\sqrt[3]{a^4}$	pow(a, 4.0/3), pow(a, 4/3.0)
	pow(a, 1.33333)

$\sqrt[4]{a^3}$	pow(a, 3.0/4), pow(a, 3/4.0)
	pow(a, 0.75)

$\frac{a+b}{c}$	(a + b) / c
$\frac{a+b}{c-d}$	(a + b) / (c - d)
$\frac{a^3+b^3}{c^2-d^2}$	(pow(a,3) + pow(b,3))/(pow(c,2) - pow(d,2)) (a*a*a + b*b*b)/(c*c - d*d)
$4\pi rad^2$	4*3.14*rad*rad 4*3.14*pow(rad,2)
$\log(x)+\log(y)$	log10(x) + log10(y) logarithm to base 10 log(x) + log(y) logarithm to base e

Typecasting (Explicit Type Conversion): converting data from one type to another type temporarily, inside the processor (CPU). Examples of Type casting are given below:

```
#include<iostream.h>
void main()
{
   int m, n;
   cout<<"Input 2 integers? ";
   cin>>m>>n;
   double r1=double(m)/n;
   double r2=(double)m/n;
   cout<<r1<<','<<r2<<endl;
}
```

This program displays the result of integer `m` divided by integer `n`. If `m` is a multiple of `n` then the result will be perfect. But if `m` is not divisible by `n`, then the result will be the integer part, fractional part will be lost. By typecasting variable `m` to **`double`**, will give a floating point output.

```
#include<iostream.h>
void main()
{
   char ch;
   cout<<"Character? "; cin>>ch;
   int code=int(ch);
   cout<<"ASCII Code="<<code<<endl;
}
```

This program displays the ASCII code of an inputted character.
`int code=int (ch);`
On the right hand side, variable `ch` is temporarily converted to int inside the CPU. The ASCII code of `ch` is stored in the variable `code`. In this example the parenthesis is around the data type.

Automatic (Implicit) Type Conversion: converting data type of the variable / expression / constant which is on the right hand side of assignment operator, to the data type of the variable which is on the left hand of the assignment operator.

```
#include<iostream.h>
void main()
{
   char ch=65;
   int code='Z';
   cout<<ch<<","<<code<<endl;
}
```

This program displays `A` and `97`. Integer constant `65` is automatically converted to **`char`** type. Character constant `'Z'` is automatically converted to **`int`** type.

Character Data

A standard group of letters, digits, punctuation marks and control characters used by computer represent character type data. PC uses **ASCII** (**A**merican **S**tandard **C**ode **F**or **I**nformation **I**nterchange) character set. ASCII character set consists of 256 characters.

ASCII code: Every character in ASCII character set is assigned a unique integer value starting from 0. First character in the ASCII character set is assigned a value 0 (zero) and 256th character is assigned a value 255. Characters with ASCII codes 31 to 127 are displayed below:

□	32	!	33	"	34	#	35	$	36	%	37	&	38	'	39	(	40	)	41
*	42	+	43	,	44	-	45	.	46	/	47	0	48	1	49	2	50	3	51
4	52	5	53	6	54	7	55	8	56	9	57	:	58	;	59	<	60	=	61
>	62	?	63	@	64	A	65	B	66	C	67	D	68	E	69	F	70	G	71
H	72	I	73	J	74	K	75	L	76	M	77	N	78	O	79	P	80	Q	81
R	82	S	83	T	84	U	85	V	86	W	87	X	88	Y	89	Z	90	[	91
\	92	]	93	^	94	_	95	`	96	a	97	b	98	c	99	d	100	e	101
f	102	g	103	h	104	i	105	j	106	k	107	l	108	m	109	n	110	o	111
p	112	q	113	r	114	s	115	t	116	u	117	v	118	w	119	x	120	y	121
z	122	{	123	|	124	}	125	~	126	DEL	127								

Extended ASCII character set (Characters with ASCII codes 31 to 127) is given below:

Ç 128	ü 129	é 130	â 131	ä 132	à 133	å 134	ç 135	ê 136	ë 137
è 138	ï 139	î 140	ì 141	Ä 142	Å 143	É 144	æ 145	Æ 146	ô 147
ö 148	ò 149	û 150	ù 151	ÿ 152	Ö 153	Ü 154	¢ 155	£ 156	¥ 157
₧ 158	ƒ 159	á 160	í 161	ó 162	ú 163	ñ 164	Ñ 165	ª 166	º 167
¿ 168	⌐ 169	¬ 170	½ 171	¼ 172	¡ 173	« 174	» 175	░ 176	▒ 177
▓ 178	│ 179	┤ 180	╡ 181	╢ 182	╖ 183	╕ 184	╣ 185	║ 186	╗ 187
╝ 188	╜ 189	╛ 190	┐ 191	└ 192	┴ 193	┬ 194	├ 195	─ 196	┼ 197
╞ 198	╟ 199	╚ 200	╔ 201	╩ 202	╦ 203	╠ 204	═ 205	╬ 206	╧ 207
╨ 208	╤ 209	╥ 210	╙ 211	╘ 212	╒ 213	╓ 214	╫ 215	╪ 216	┘ 217
┌ 218	█ 219	▄ 220	▌ 221	▐ 222	▀ 223	α 224	ß 225	Γ 226	π 227
Σ 228	σ 229	µ 230	τ 231	Φ 232	Θ 233	Ω 234	δ 235	∞ 236	φ 237
ε 238	∩ 239	≡ 240	± 241	≥ 242	≤ 243	⌠ 244	⌡ 245	÷ 246	≈ 247
° 248	∙ 249	· 250	√ 251	ⁿ 252	² 253	■ 254	255		

Type Modifier

Type modifiers are used to change default type of the built-in data types. Type modifiers supported by C++ are **`long`**, **`short`**, **`signed`** and **`unsigned`**.

Data type	Storage	Range
`char`	1 byte / 8 bits	-128 … 127
`int`	4 bytes / 32 bits	-2147483648 … 2147483647
`float`	4 bytes / 32 bits	$3.4*10^{-38}$ … $3.4*10^{38}$
`double`	8 bytes / 64bits	$1.7*10^{-308}$ … $1.7*10^{308}$
`unsigned char`	1 byte / 8 bits	0 … 255
`signed char`	1 byte / 8 bits	-128 … 127
`short int`	2 bytes / 16 bits	-32768 … 32767
`long int`	4 bytes / 16 bits	-2147483648 … 2147483647
`signed int`	4 bytes / 32 bits	-2147483648 … 2147483647
`unsigned int`	4 bytes / 32 bits	0 … 4294967295
`short signed int`	2 bytes / 16 bits	-32768 … 32767
`short unsigned int`	2 bytes / 16 bits	0 … 65535
`long signed int`	4 bytes / 32 bits	-2147483648 … 2147483647
`long unsigned int`	4 bytes / 32 bits	0 … 4294967295
`short`	2 bytes / 16 bits	-32768 … 32767
`long`	4 bytes / 32 bits	-2147483648 … 2147483647
`signed`	4 bytes / 32 bits	-2147483648 … 2147483647
`unsigned`	4 bytes / 32 bits	0 … 4294967295
`long double`	10 bytes / 80 bits	$3.4*10^{-4932}$ … $1.1*10^{4932}$

- **`void`** and **`float`** does not support any type modifiers
- **`int`** supports all the four type modifiers, **`char`** supports **`signed`** and **`unsigned`** and **`double`** supports only **`long`**.
- Type modifiers are used fundamental data type to create a variable. But if a variable is created using only type modifier then the default data type for the variable is **`int`**

Block (Compound Statement):

C++ statement(s) written within a pair of braces ({}) is called a block. A block may contain one or more statements. A block may not contain any statement, that is, a block may be empty.

Token

Building block of a program is called a token. It is also called program element. Tokens of a C++ program can be classified as Keyword, Identifier, Constant, Operator, String and Comment.

a) **Keyword**: It is component of a program which has special meaning for the C++ compiler. In Borland C++ editor keyword appear in **bold** face. C++ compiler contains list of all the keywords. List of keywords vary from to compiler to compiler. **A keyword cannot be redefined**. List of commonly used C++ keywords are given below:

break	case	char	class	const
continue	default	delete	do	double
else	enum	extern	float	for
friend	goto	huge	if	inline
int	long	new	operator	private
protected	public	register	return	short
signed	sizeof	static	struct	switch
this	throw	try	typedef	union
unsigned	using	virtual	void	while

Highlighted keywords are listed in Computer Science Syllabus.

b) **Identifier**: Identifier is a component of a program which is identified by a C++ compiler. There are two broad categories of identifiers:

Built-in: It is name of built-in functions, constants, variables, classes and structure. To use built-in identifier we need appropriate header file. **Built-in identifier can be redefined**.

User-defined: Name created by the programmer like variable names, user-defined function names, constant names, class names and structure names. User-defined identifiers can only be used after they have created or declared.

c) **Constant**: A constant is a program element whose value remains same through the program. Examples of different types of constants are given below:

Data Type	Constants
char	'A', 'B', 't', 'x', '0' , '6', '9', '*', '+', '['
int	4, 10, 169, 1234, 0, -71238, -1025, -45, 331, -5
double	0.0, -2.3333, 15.75, -154.85, 96.625, 1.25, -7.8

d) **Operator**: Operators are used in C++ to carry out various functions. Mostly operators are used in arithmetic calculations and in logical expressions. But operators may be used for dynamic memory management. An operator in C++ can be **unary**, **binary** and **ternary**. Examples of operators are given below:

Operator	Expression	Meaning
unary +	+ a	Sign of value stored in a remains unaltered
unary -	- a	Changes sign of value stored in a
Binary +	a + b	Adds a and b
Binary -	a – b	Subtract b from a
*	a * b	Multiply a and b
/	a / b	Divide a by b
%	a % b	Remainder of a divided by b
=	a = 10	a is assigned a value 10
++	++a, a++	Increments value stored in a by 1
--	--a, a--	Decrements value stored in a by 1
+=	a += b	b is added to a and the result is assigned a
-=	a -= b	b is subtracted from a and the result is assigned a

*=	a *= b	a is multiplied by b and the result is assigned a
/=	a /= b	a is divided by b and the result is assigned a
%=	a %= b	a is assigned a value of a % b
==	a == b	a is equal to b
!=	a != b	a is not equal to b
>	a > b	a is greater than b
>=	a >= b	a is greater than equal to b
<	a < b	a is less than b
<=	a <= b	a is less than equal to b
!	!(a < b)	Negate the condition a is less than b
&&	a>10 && a<20	a's value lies between 10 and 20.
||	a<10 || a>20	a's value is either less than 10 or greater than 20

Unary operator: An operator that needs **one operand**.
Examples: Unary -, unary +, ++, -- and !.

Binary operator: An operator that needs **two operands**.
Example: Binary +, Binary -, *, /, %, C++ short hand operators, logical operators, && and ||.

Ternary operator: An operator that needs **three operands**. Ternary operator is also known as Conditional operator. Relational operators (>, >=, <, <=, ==, !=), Logical operators (!, &&, ||) and Ternary operator (?:) will be discussed with **if-else** statement.

Operators working form Left to Right	Operators working form Right to Left
+, -, *, /, %, >, >=, >, <=, ==, !=, &&, ||, !, ++, --	=, +=, -=, *=, /=, %=

e) **String**: In C++ anything enclosed within a pair of double quotes (") is a called a String constant. A string is treated as an array of character or as a pointer to a character. Array and pointer will be discussed later. Examples of string are given below:

```
"India", "35/8", "999", "***", "GH-14/200", "6", "A", "#", ""
```

f) **Comment**: Non executable statements of a C++ program are called Comments. Comments are also known as Remarks. A Comment is completely ignored by a compiler. No code is generated for a Comment. Comment is a good tool for Debugging. C++ supports two types of Comments:

Single line Comment: also known as C++ style Comments. Single Line Comment starts with pair of forward slash (//) and till the end of line is considered as a Comment. Examples of Single Line Comment are given below:

```
// single line comment
// comment in C++ style
```

Multi-line comment: also known as C style comments. Multi-line comment start with forward slash and star (/*) and with star and forward slash (*/). Examples of Multi-Line Comment are given below:

```
/*
  multi-line comments
  comment in C style
*/
/* Single line comment */
```

Compiler directive: instruction given to the compiler. Compiler directive is also called Pre-processor. C++ statement is an instruction given to CPU or to the computer. It is called Pre-Processor because instruction to the compiler given before the processing starts. Every Compiler Directive begins with hash (#). Examples of Compiler Directives are given below:

`#include`:	to include header files
`#define`:	to create C++ macros

C++ Shorthand: C++ allows an expression to be written in a compact form. C++ shorthand works with character (**`char`**) type data, integer (**`int`**) type data and floating point (**`float`** and **`double`**) type data. Examples of C++ shorthand are given below:

Operator	Expression	Expansion	Meaning
`+=`	`a += b`	`a = a + b`	Variable `a` is assigned a value `a + b`
`-=`	`a -= b`	`a = a - b`	Variable `a` is assigned a value `a - b`
`*=`	`a *= b`	`a = a * b`	Variable `a` is assigned a value `a * b`
`/=`	`a /= b`	`a = a / b`	Variable `a` is assigned a value `a / b`
`%=`	`a %= b`	`a = a % b`	Variable `a` is assigned a value `a % b`

```
#include<iostream.h>
void main()
{
   int a=5, b=7;
   b+=a;
   a*=b;
   cout<<a<<','<<b<<endl;
   a/=b;
   b-=a;
   cout<<a<<','<<b<<endl;
}
```

Execution of the program produces output screen

```
60,12
5,7
```

Increment Operator: Increment operator (++) increments value stored in a variable by 1 (One). Increment operator works with character (**`char`**) type data, integer (**`int`**) type data and floating point (**`float`** and **`double`**) type data. Examples of Increment operators are given below:

```
int a=10;
++a;
cout<<"Value in a="<<a<<endl;
a++;
cout<<"Value in a="<<a<<endl;
```

`++a` is Pre-increment
Increments value of a by `1`, `a`'s value is `11`
`a++` is Post-increment
Increments value of a by `1`, `a`'s value is `12`

Produces output like

```
Value in a=11
Value in a=12
```

An integer variable `x` contains `6`. The difference between pre-increment and post-increment is in the value of the expression. Table is given in the next page to explain the concept:

Operator	C++ Statement	Output	Explanation

++	`cout<<++x<<endl;`
	`cout<<x<<endl;`
		7
		7
			Increments x and then displays x
			Displays incremented values stored in x

++	`cout<<x++<<endl;`
	`cout<<x<<endl;`
		6
		7
			Displays x and then increments x
			Displays incremented values stored in x

Decrement Operator: Decrement operator (`--`) decrements value stored in a variable by 1 (One). Decrement operator works with character (**`char`**) type data, integer (**`int`**) type data and floating point (**`float`** and **`double`**) type data. Examples of Decrement operators are given below:

```
int a=7;
--a;
cout<<"Value in a="<<a<<endl;
a--;
cout<<"Value in a="<<a<<endl;
```

`--a` is Pre-decrement
Decrements value of a by `1`, `a`'s value is `6`
`a--` is Post-deccrement
Decrements value of a by `1`, `a`'s value is `5`

Produces output like

```
Value in a=6
Value in a=5
```

Let us assume that an integer variable `z` contains a value `26`. The table given below displays the difference between pre-decrement operator and post-decrement operator.

Operator	C++ Statement	Output	Explanation

--	`cout<<--z<<endl;`
	`cout<<z<<endl;`
		25
		25
			Decrements z and then displays z
			Displays decremented values stored in z

--	`cout<<z--<<endl;`
	`cout<<z<<endl;`
		26
		25
			Displays z and then decrements z
			Displays decremented values stored in z

String: as mentioned earlier string is not a fundamental data type. String is an array of characters (derived data type). To create a string variable we need to do the following:

Rule: **`char`** `strvar[size];`
Usage

```
char name[20];
char address[80];
```

`strvar` is the name of the string variable and `size` is a positive integer constant representing maximum number of characters that can be stored under `strvar` name. If the string size is 20, then actually we can store maximum 19 characters and one place for the nul character. We can use `cin` to input a string value into a string variable. An example is given below:

```
#include<iostream.h>
void main()
{
   char city[20];
   cout<<"City Name? "; cin>>city;
   cout<<"City="<<city;
}
```

First run of program produces following screen:

```
City? Kolkata
City=Kolkata
```

Second run of program produces following screen:

```
City? New Delhi
City=New
```

Using `cin` we cannot input a string that contains space/tab. To input a string with space/tab, we have to use function `gets`() from the header file <stdio.h>.

Modified program with `gets`() is given below:

```
#include<iostream.h>
#include<stdio.h>
void main()
{
   char city[20];
   cout<<"City Name? "; gets(city);
   cout<<"City="<<city;
}
```

First run of program produces following screen:

```
City? Kolkata
City=Kolkata
```

Second run of program produces following screen:

```
City? New Delhi
City=New Delhi
```

Syntax error: An error in a C++ statement when the rule (grammar / syntax) of the programming language is violated. Examples of Syntax errors are given below:

a) Typographical mistakes
b) Omitted semicolons or coma
c) References to undeclared variables
d) Wrong number or type of parameters passed to a function

Syntax errors are detected by the compiler. Syntax errors are also known as **Compile-Time** errors because the errors are detected by the compiler during compilation time.

Run-time error: Syntactically correct statement performs illegal operation during execution of a program. Illegal operation is performed when the program encounters an unexpected data. Examples of Run-Time errors are given below:

a) Division by zero (0)
b) Square root of a negative number
c) Logarithm of zero (0) or negative number

Logical error: Syntactically correct statement produces unexpected output due to an error in program design or flaw in the program implementation. Logical error may trigger a run-time error. Examples of Logical errors are given below:

a) Incorrect placement of braces (curly brackets) for a block
b) Missing parenthesis when parenthesis is required
c) Mixing up = (assignment operator) with == (equal operator)
d) Accumulator or counter not initialized

1. Write a complete C++ program to input temperature in Celsius and convert it into Fahrenheit. Display temperature in Fahrenheit on the screen.

```
#include<iostream.h>
void main()
{
   double tcel;
   cout<<"Temperature in Celsius? "; cin>>tcel;
   double tfar=1.8*tcel+32;
   cout<<"Temperature in Celsius="<<tfar<<endl;
   cout<<"Temperature in Fahrenheit="<<tfar<<endl;
}
```

2. Write a complete C++ program to input temperature in Fahrenheit and convert it into Celsius. Display temperature in Celsius on the screen.

```
#include<iostream.h>
void main()
{
   double tfar;
   cout<<"Temperature in Fahrenheit? "; cin>>tfar;
   double tcel=(tfar-32)/1.8;
   cout<<"Temperature in Fahrenheit="<<tfar<<endl;
   cout<<"Temperature in Celsius="<<tfar<<endl;
}
```

2. Write a complete C++ program to input radius of a circle; calculate area and circumference of the circle. Display area and circumference on the screen.

```
#include<iostream.h>
void main()
{
   double rad;
   cout<<"Input radius? "; cin>>rad;
   double area=3.14*rad*rad, circum=2*3.14*rad;
   cout<<"Area         ="<<area<<endl;
   cout<<"Circumference="<<circum<<endl;
}
```

4. Write a complete C++ program to input radius of a sphere; calculate surface area and volume of the sphere. Display surface area and volume on the screen.

```
#include<iostream.h>
void main()
{
   double rad;
   cout<<"Input radius? "; cin>>rad;
   double sar=4*3.14*rad*rad, vol=4/3.0*3.14*rad*rad*rad;
   cout<<"Suface Area="<<sar<<endl;
   cout<<"Volume     ="<<vol<<endl;
}
```

5. Write a complete C++ program to input radius and height of a solid cylinder; calculate surface area and volume of the solid cylinder. Display surface area and volume on the screen.

```
#include<iostream.h>
void main()
{
  double rad, ht;
  cout<<"Input radius? "; cin>>rad;
  cout<<"Input height? "; cin>>ht;
  double sar=2*3.14*rad*(rad+ht);
  double vol=3.14*rad*rad*ht;
  cout<<"Radius     ="<<rad<<endl;
  cout<<"Height     ="<<ht<<endl;
  cout<<"Suface Area="<<sar<<endl;
  cout<<"Volume     ="<<vol<<endl;
}
```

6. Write a complete C++ program to input base and height of a triangle; calculate area of a triangle. Display area of the triangle on the screen.

```
#include<iostream.h>
void main()
{
  double base, ht;
  cout<<"Input base  ? "; cin>>base;
  cout<<"Input height? "; cin>>ht;
  double area=0.5*base*ht;
  cout<<"Base            ="<<area<<endl;
  cout<<"Height          ="<<ht<<endl;
  cout<<"Area of Triangle="<<area<<endl;
}
```

7. Write a complete C++ program to input length of three side of a triangle; calculate area of a triangle using Heron’s formula. Display area of the triangle on the screen.

```
#include<iostream.h>
#include<math.h>
void main()
{
  double a, b, c;
  cout<<"Length of 1st side? "; cin>>a;
  cout<<"Length of 2nd side? "; cin>>b;
  cout<<"Length of 3rd side? "; cin>>c;
  double s=(a+b+c)/2;
  double area=sqrt(s*(s-a)*(s-b)*(s-c));
  cout<<"Length of 1st side="<<a<<endl;
  cout<<"Length of 2nd side="<<b<<endl;
  cout<<"Length of 3rd side="<<c<<endl;
  cout<<"Area of the triangle="<<area<<endl;
}
```

8. Write a complete C++ program to input 3 coefficient of a quadratic equation ($ax^2+bx+c=0$); calculates two roots of the quadratic equation. Display two roots on the screen.

```
#include<iostream.h>
#include<math.h>
void main()
{
   double a, b, c;
   cout<<"Coefficient of x^2? "; cin>>a;
   cout<<"Coefficient of x  ? "; cin>>b;
   cout<<"Constant Term     ? "; cin>>c;
   double d=b*b-4*a*c;
   double x1=(-b+sqrt(d))/(2*a), x2=(-b-sqrt(d))/(2*a);
   cout<<"x1="<<x1<<endl;
   cout<<"x2="<<x2<<endl;
}
```

9. Write a complete C++ program to input roll (integer), student name (string), theory marks (floating point: out of 70), practical marks (floating point: out of 30) and weekly test marks (floating point: out of 40); calculate term total (floating point: theory marks + practical marks) and grand total (floating point: 80% of term total + 50% of weekly test). Display roll, student name, theory marks, practical marks, weekly test marks, term total and grand total.

```
#include<iostream.h>
#include<stdio.h>
void main()
{
   int roll;
   char name[20];
   double theo, prac, wtest;
   cout<<"Roll Number? "; cin>>roll;
   cout<<"Student Name? "; gets(name);
   cout<<"Theory marks[0-70]? "; cin>>theo;
   cout<<"Practical marks[0-30]? "; cin>>prac;
   cout<<"Weekly Test marks[0-40]? "; cin>>wtest;
   double term=theo+prac, gtot=0.8*term+0.5*wtest;
   cout<<"Roll Number="<<roll<<endl;
   cout<<"Name       ="<<name<<endl;
   cout<<"Theory     ="<<theo<<endl;
   cout<<"Practical  ="<<prac<<endl;
   cout<<"Term Total ="<<term<<endl;
   cout<<"Weekly Test="<<wtest<<endl;
   cout<<"Grand Total="<<gtot<<endl;
}
```

10. Write a complete C++ program to input employee code (integer); employee name (string), basic (floating point); calculate house rent (floating point: 40% of basic), dearness allowance (floating point: 65% of basic), city allowance (floating point: 15% of basic), gross salary (floating point: basic + house rent + dearness allowance + city allowance), provident fund deductions (floating point: 10% of gross salary) and net salary (floating point: gross salary - provident fund deductions). Display employee code, employee name, basic, house rent, dearness allowance, city allowance, gross salary, provident fund deductions and net salary.

```
#include<iostream.h>
#include<stdio.h>
void main()
{
   int code;
   char name[20];
   double basic;
   cout<<"Employee Code? "; cin>>code;
   cout<<"Employee Name? "; gets(name);
   cout<<"Basic Salary ? "; cin>>basic;
   double hrent=0.4*basic;
   double dallow=0.65*basic;
   double callow=0.15*basic;
   double gross=basic+hrent+dallow+callow;
   double pfund=0.1*gross;
   double net=gross-pfund;
   cout<<"Employee Code      ="<<code<<endl;
   cout<<"Employee Name      ="<<name<<endl;
   cout<<"Basic Salary       ="<<basic<<endl;
   cout<<"House Rent         ="<<hrent<<endl;
   cout<<"Dearness Allowance ="<<dallow<<endl;
   cout<<"City Allowance     ="<<callow<<endl;
   cout<<"Gross Salary       ="<<gross<<endl;
   cout<<"Provident Fund     ="<<pfund<<endl;
   cout<<"Net Salary         ="<<net<<endl;
}
```