

Chapter-7

INTRODUCTION TO C++

➤ History of C++:

- Until 1980, C programming was widely popular, and slowly people started realizing the drawbacks of this language, at the same time a new programming approach that was Object Oriented Programming.
- The C++ programming language was created by *Bjarne Stroustrup* and his team at Bell Laboratories (AT&T, USA) to help implement simulation projects in an object-oriented and efficient way.
- C++ is a superset of C because; any valid C program is valid C++ program too but not the vice versa is not true.
- C++ can make use of existing C software libraries with major addition of “Class Construct”.
- This language was called “C with classes” and later in 1983, it was named “C++” by Rick Mascitii.
- As the name C++ implies, C++ was derived from the C programming language: ++ is the increment operator in C.

➤ Characteristics of C++:

- **Object-Oriented Programming:** It allows the programmer to design applications like a communication between object rather than on a structured sequence of code. It allows a greater reusability of code in a more logical and productive way.
- **Portability:** We can compile the same C++ code in almost any type of computer & operating system without making any changes.
- **Modular Programming:** An application’s body in C++ can be made up of several source code files that are compiled separately and then linked together saving time.
- **C Compatibility:** Any code written in C can easily be included in a C++ program without making any changes.
- **Speed:** The resulting code from a C++ compilation is very efficient due to its duality as high-level and low-level language.
- **Machine independent:** It is a Machine Independent Language.
- **Flexibility:** It is highly flexible language and versatility.
- **Wide range of library functions:** It has huge library functions; it reduces the code development time and also reduces cost of software development.
- **System Software Development:** It can be used for developing System Software Viz., Operating system, Compilers, Editors and Database.

➤ C++ Character Set:

- Character Set means the valid set of characters that a language can recognizes.
- The character set of C++ includes the following:

Alphabets	Upper letters case A, B, C, D.....X, Y, Z		
	Lower letters case a, b, c, d.....x, y, z		
Digits	0,1,2,3.....9		
Special Characters	, comma	. Period	` Apostrophe
	: Colon	; Semicolon	? Question mark
	! Exclamation	_ Underscore	Pipeline
	{ Left brace	} Right Brace	# Hash
	[Left bracket] Right Bracket	^ Caret
	(Left parenthesis) Right parenthesis	& ampersand
	/ Slash	\ Back slash	~ Tilde
	+ Plus sign	- Minus Sign	< Less Than
* Asterisk	% Percentage	> Greater Than	

➤ C++ Tokens:

- *The smallest individual unit in a program is known as **token**.*
- These elements help us to construct statements, definitions, declarations, and so on, which in turn helps us to construct complete program.
- Tokens used in C++ are:
 1. Identifier
 2. Reserved Keywords
 3. Constants or Literals
 4. Punctuators
 5. Operators

➤ Identifiers:

- Identifiers is a name given to programming elements such as variables, functions, arrays, objects, classes, etc.
- It contains letters, digits and underscore.
- C++ is a case sensitive; it treats uppercase and lowercase characters differently.
- The following are some valid identifiers: Pen time580 s2e2r3 _dos _HJI3_JK

- **Rules to be followed while creating identifiers:**

- Identifiers are a sequence of characters which should begin with the alphabet either from A-Z (uppercase) or a-z (lowercase) or _ (underscore).
- C++ treats uppercase and lowercase characters differently. For example, DATA is not same as data.
- No Special character is allowed except underscore “_”.
- Identifier should be single words i.e. blank spaces cannot be included in identifier.
- Reserved Keywords should not be used as identifiers.
- Identifiers should be of reasonable length.

- **Keywords:**

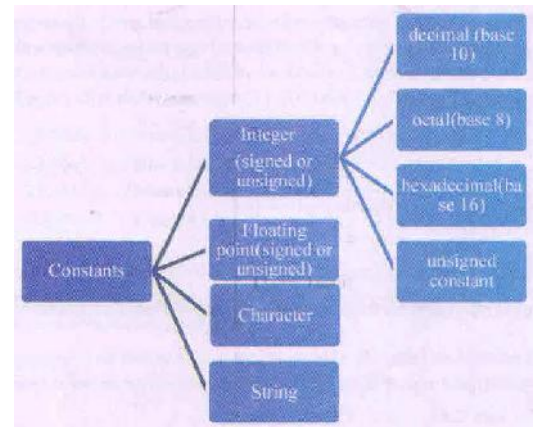
- Keyword is a predefined word that gives special meaning to the compiler. The programmer is not allowed to change its meaning.
- These are reserve for special purpose and must not be used as identifier name.
- Example: for, if, else, this, do, float, while, switch etc.
- There are keywords in C++ as mentioned below:

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

- **Constants:**

- A constant are identifiers whose value does not change during program execution.
- Constants are sometimes referred to as literal
- A constant or literal my be any one of the following:
 - Integer Constant

- Floating Constant
- Character Constant
- String Constant



✓ Integer Constant:

- An integer constant is a whole number which can be either positive or negative.
- They do not have fractional part or exponents.
- We can specify integer constants in decimal, octal or hexadecimal form.

- **Decimal Integer Constant:** It consists of any combination of digits taken from the set 0 to 9.

For example:

```

int a = 100;           //Decimal Constant
int b = -145          // A negative decimal constant
int c = 065           // Leading zero specifies octal constant, not decimal
  
```

- **Octal Integer Constant:** It consists of any combination of digits taken from the set 0 to 7. However the first digit must be 0, in order to identify the constant as octal number.

For example:

```

int a = 0374;         //Octal Constant
int b = 097;          // Error: 9 is not an octal digit.
  
```

- **Hexadecimal Integer Constant:** A Sequence of digits begin the specification with 0X or 0x, followed by a sequence of digits in the range 0 to 9 and A (a) to F(f).

For example:

```

int a = 0x34;
int b = -0XABF;
  
```

- **Unsigned Constant:** To specify an unsigned type, use either u or U suffix. To specify a long type, use either the l or L suffix.

For example:

```

unsigned    a = 328u;           //Unsigned value
long        b = 0x7FFFFFFL;     //Long value specified as hex constant
unsigned long c = 0776745ul;    //Unsigned long values as octal constant
  
```

✓ Floating Point Constant:

- Floating point constants are also called as “**real constants**”.
- These values contain decimal points (.) and can contain exponents.

- They are used to represent values that will have a fractional part and can be represented in two forms (i.e. fractional form and exponent form)
 - Floating-point constants have a “mantissa”, which specifies the value of the number, an “exponent” which specifies the magnitude of the number, and an optional suffix that specifies the constant’s type.
 - The mantissa is specified as a sequence of digits followed by a period, followed by an optional sequence of digits representing the fractional part of the number.
 - The exponent, if present, specifies the magnitude of the number as a power of 10.
 - Example: `23.46e0` // means it is equal to $23.46 \times 10^0 = 23.46 \times 1 = 23.46$
 - It may be a positive or negative number. A number with no sign is assumed to be a positive number. For example, `345.89`, `3.142`
- ✓ **Character Constants:**
- Character constants are specified as single character enclosed in pair of single quotation marks.
 - For example `char ch = 'P';` //Specifies normal character constant
 - A single character constant such as ‘D’ or ‘r’ will have char data type. These character constants will be assigned numerical values.
 - The numerical values are the ASCII values which are numbered sequentially for both uppercase and lowercase letters.
 - For example, ASCII value of A is 65, B is 66,Z is 90 (uppercase), a is 97, b is 98..... Z is 122 (lowercase), 0 is 48, 1 is 49, 9 is 57 (digits).
 - There are certain characters used in C++ which represents character constants. These constants start with a **back slash (\)** followed by a character. They are normally called as **escape sequence**. Some of the commonly used escape sequences are.

Escape Sequence	Meaning	Escape Sequence	Meaning
<code>\'</code>	Single Quote	<code>\"</code>	Double Quote
<code>\?</code>	Question Mark	<code>\\</code>	Back Slash
<code>\0</code>	Null Character	<code>\a</code>	Audible Bell
<code>\b</code>	Backspace	<code>\f</code>	New Page
<code>\n</code>	New Line	<code>\r</code>	Carriage Return
<code>\t</code>	Horizontal Tab	<code>\v</code>	Vertical Tab
<code>\nnn</code>	Arbitrary octal value	<code>\xnn</code>	Arbitrary Hexa Value

- Escape Sequence is a special string used to control output on the monitor and they are represented by a single character and hence occupy one byte.

✓ **String Constants:**

- A string constant consists of zero or more character enclosed by double quotation marks (“”).
- Multiple character constants are called string constants and they are treated as an array of char.
- By default compiler adds a special character called the “*Null Character*” (\0) at the end of the string to mark the end of the string.
- For example: char str[15] = “C++ Programming” ;
- This is actually represented as char str[15] = “C++ Programming\0” in the memory.

➤ **Punctuators:**

- Punctuators in C++ have syntactic and semantic meaning to the compiler.
- Some punctuators can be either alone or in combination.
- The following characters are used as punctuators which are also known as separators in C++.

Punctuator	Name	Function
!	Exclamation	Used along with “=” to indicate “not equal to”
%	Percentage	Used along with format specifiers
&	Ampersand	Used to represent address location or bitwise operation
;	Semicolon	Used to represent statement terminator.
[]	Brackets	Used to array subscripts
()	Parenthesis	Used to represent function calls and function parameters.
{ }	Braces	Used to represent start and end of a block of code.
#	Ash sign	Used to represent preprocessor directives.
\	Back slash	Used to represent escape sequence
:	Colon	Used to represent labeled statement
=	Equal to	Used to represent an assigning operator.

➤ **C++ Operators:**

- *An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.*
- C++ is rich in built-in operators and there are almost 45 different operators.
- Operators in C++ are can be divided into the following classes:
 - Arithmetic Operator
 - Relational Operator
 - Logical Operator
 - Unary Operator
 - Conditional Operator
 - Bitwise Operator
 - Assignment Operator
 - Other Operator

- Operator operates on constants and variables which are called **operands**. Operators may also be classified on the number of operands they act on either:

Unary	Binary	Ternary
Unary operators operate on only one operand.	The binary operator operates on two operands.	The ternary operator operates on three operands.
Example: ++, --	+, -, *, /, %, &&,	?:

✓ Unary Operators

- Unary operators have only one operand*; they are evaluated before any other operation containing them gets evaluated.
- The following are the list of unary operators.

Operator	Name	Function
!	Logical NOT	If a condition is true then Logical NOT operator will make false.
&	Address-of	Used to give the address of the operand
~	One's Complement	Converts 1 to 0 and 0 to 1
*	Pointer dereference	Used along with the operand to represent the pointer data type.
+	Unary plus	Used to represent a signed positive operand
++	Increment	Used to increment an operand by 1
-	Unary negation	Used to represent a signed negative operand
--	Decrement	Used to represent an operand by 1

◆ Increment Operator


Increment operator is used to increasing the value of an integer by one. This is represented by “++”.

Example: a++, a+1

◆ Decrement Operator

Decrement operator is used to decreasing the value of an integer by one. This is represented by “--”.

Example: a--, a-1

 Hint	Let a=10 and b=5 a++; //a becomes 11 b--; //b becomes 4
--	---

✓ **Both the increment & decrement operators come in two versions:**

Prefix increment/decrement:

- When an increment or decrement operator precedes its operand, it is called prefix increment or decrement (or pre-increment / decrement).
- In prefix increment/decrement, C++ performs the increment or decrement operation before using the value of the operand.

- Example: If sum = 10 and count =20 then

Sum = sum + (++count);

- First count incremented and then evaluate sum = 31.

Postfix increment/decrement:

- When an increment or decrement operator follows its operand, it is called postfix increment or decrement (or post-increment / decrement).
- In postfix increment/decrement, C++ first uses the value of the operand in evaluating the expression before incrementing or decrementing the operand's value.

- Example: If sum = 10 and count =20 then

Sum = sum + (count++);

- First evaluate sum = 30, and then increment count to 21.

✓ **Binary Operators**

- The binary operators are those operators that operate on two operands. They are as arithmetic, relational, logical, bitwise, and assignment operators.

✓ **Arithmetic Operator**

- Arithmetic operators are used to performing the basic arithmetic operations such as arithmetic, subtraction, multiplication, division and modulo division (remainder after division).

Operator	Description	Example(a=10, b=20)
+	Adds two operand	a + b = 30
-	Subtracts second operand from the first	a - b = -10
*	Multiply both operand	a * b = 200
/	Divide numerator by denominators	b / a = 2
%	Modulus operators and remainder of after an integer division	b % a = 0

✓ Relational Operator

- Relational Operator is used to comparing two operands given in expressions.
- They define the relationship that exists between two constants.
- For example, we may compare the age of two persons or the price of two items....these comparisons can be done with the help of relational operators.
- The result in either TRUE(1) or FALSE(0).Some of the relational operators are:

Operator	Description	Example (a=10, b=5)
<	Checks if the value of left operand is less than the value of right operand	a < b returns false(0)
<=	Checks if the value of left operand is less than or equal to the value of right operand	a <= b returns false(0)
>	Checks if the value of left operand is greater than the value of right operand	a > b returns true(1)
>=	Checks if the value of left operand is greater than or equal to the value of right operand	a >= b returns false(0)
==	Checks if the value of two operands is equal or not	a == b returns false(0)
!=	Checks if the value of two operands is equal or not	a != b returns true(1)

✓ Logical Operators

- Logical operators are used to testing more than one condition and make decisions. Some of the logical operators are

Operator	Meaning	Description	Example
&&	Logical AND	If both the operands are non-zero then condition becomes true.	If a=10 and b=5 then, ((a==10) && (b>5)) returns false.
	Logical OR	If any of the two operands is non-zero then condition becomes true.	If a=10 and b=5 then, ((a==10) (b>5)) returns true.
!	Logical NOT	If a condition is true then the Logical NOT operator will make false.	If a=10 then, !(a==10) returns false.

✓ Bitwise Operators

- A bitwise operator works on bits and performs bit by bit operation.
- Bitwise operators are used in bit level programming.

Operators	Meaning of operators
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement

- The truth table for bitwise AND (&), Bitwise OR(|), Bitwise XOR (^) are as follows:

A	B	A&B (Bitwise AND)	A B (Bitwise OR)	A ^ B (Bitwise XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Assume A = 60 and B =13; the following operations take place:

Step 1: Converts A and B both to its binary equivalent.

$$A = 0011\ 1100$$

$$B = 0000\ 1101$$

Step 2: Then performs the bitwise and, or and not operation. The result is given below.

$$A \& B = 0000\ 1100 = 12$$

$$A | B = 0011\ 1101 = 61$$

$$A \wedge B = 0011\ 0001 = 49$$

$$\sim A = 1100\ 0011 = -60$$

The Bitwise operators supported by C++ are listed in the following table:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands	(A&B) will give 12 which is 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A^B) will give 49 which is 0011 0001
~	Binary Ones complement Operator is unary and has the effect of 'Flipping' bits	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right	A<<2 will give 240 which is 1111 0000

	operand.	
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A>>2 will give 15 which is 0000 1111

✓ Assignment Operators

- The most common assignment operator is =. This operator assigns the value on the right side to the left side.
- Example:**

```
var = 5 //5 is assigned to var
a = b; //value of b is assigned to a
5 = b; // Error! 5 is a constant.
```

The assignment operators supported by C++ are listed below:

Operator	Example	Same as
=	a=b	a=b
+=	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
<<=	a<<=2	a = a<<2
>>=	a>>=2	a = a>>2
&=	a & = 2	a = a & 2
^=	a ^ = 2	a = a ^ 2
=	a = 2	a = a 2

✓ C++ Shorthand's:

- C++ Offers special shorthand's that simplify the coding of a certain type of assignment statements.
- The general format of C++ shorthand's is:
- Variable Operator = Expression
- Following are some examples of C++ shorthand's:

x - = 10;	Equivalent to	x = x - 10;
x * = 5;	Equivalent to	x = x * 5;
x / = 2 ;	Equivalent to	x = x / 2;
x % = z;	Equivalent to	x = x % z;

➤ Conditional Operator:

- A ternary operator pair "?:" is available in C++ to construct conditional expressions of the form: **exp1? exp2: exp3**, where exp1, exp2, and exp3 are expressions,
- The operator "?:" works as follows: exp1 is evaluated first. If it is true, then the expression exp 2 is evaluated and becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.
- **Example:** a=10; b=5;
 x = (a>b) ? a:b;

➤ Special Operator:

Operators	Meaning of operators
sizeof()	It is a unary operator which is used in finding the size of the data type. Example: sizeof(a)
, (comma)	Comma operators are used to linking related expressions together. Example: int a=10, b=5
. (dot) and -> (arrow)	Member Operator used to reference individual members of classes, structure and unions.
cast	Casting Operator convert one data type to another.
&	Address Operator & returns the address of the variable.
*	Pointer Operator * is pointer to a variable.

✓ Precedence of Operators or Hierarchy of Operators In C++:

- An expression is a combination of opcode and operand.
- The operators would be arithmetic, relational, and logical operators.
- If the expression contains multiple operators, the order in which operations carried out is called the precedence of operators. It is also called as priority or hierarchy.
- The Operators with highest precedence appear at the top of the table and those with the lowest appear at the bottom.

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to Right
Unary	= - ! ~ ++ -- (type) * & sizeof	Right to Left
Multiplicative	* / %	Left to Right
Additive	+ -	Left to Right
Shift	<< >>	Left to Right

Relational	<<= >>=	Left to Right
Equality	== !=	Left to Right
Bitwise AND	&	Left to Right
Bitwise XOR	^	Left to Right
Bitwise OR		Left to Right
Logical AND	&&	Left to Right
Logical OR		Left to Right
Conditional	?:	Right to Left
Assignment	= += -= *= /= %=	Right to Left
Comma	,	Left to Right

What is operator precedence in C++?

- “The order in which different types of operators are evaluated is called as operator precedence”.
- It is also known as hierarchy of operators.
- In any expression, Operators having higher precedence are evaluated first.
- The expression is evaluated in the following sequence.
 - Arithmetic
 - Relational
 - Logical
- There are some operators which are given below. The higher the position of an operator is, higher is its priority.

Operator	Meaning
!	Logical NOT
()	Parenthesis
*, /, %	Arithmetic and modulus
+, -	Arithmetic
<, >, <=, >=	Relational
==, !=	Relational
&&	Logical AND
	Logical OR
=	Assignment

➤ Type Conversion:

- Converting an expression of a given type into another type is known as type-casting or type conversion.
- Type conversions are of two types, they are:
 - Implicit Conversion
 - Explicit Conversion

✓ Implicit Conversion:

- Implicit Conversions do not require any operator.
- They are automatically performed when a value is copied to a compatible type.
- The C++ compiler will implicitly convert or promote values if it can be done safely.
- If not it will generate a warning or an error depending on the conversion required.
- For Example:

```
short a = 2000;
int b;
b = a;
```

✓ Explicit Conversion:

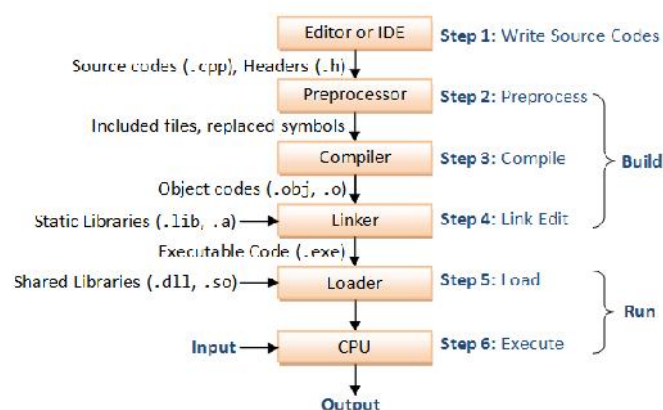
- C++ is a strong-typed language. Many conversions, especially those that imply a different interpretation of the value, require an explicit conversion.

- For Example:

```
short a = 2000;
int b;
b = (int) a;    //c-like cast notation
b = int (a)    // functional notation
```

- Implicit Conversions do not require any operator.

➤ Translating a C++ program:



➤ General Structure of C++ Program:

General Syntax	Example Program
<pre> /* General Structure */ Pre-processor Directives main () { Variable Declarations; Executable Statements; } </pre>	<pre> /* A Simple Program to display a Message */ #include<iostream.h> void main() { cout<<" This is the first C++ Program"; } </pre>

Different programming languages have their own format of coding. The basic components of a C++ program are:

❖ Comments or Documentation Section:

- Comments are the pieces of code that compiler ignores to compile. There are two types of comments in C++.
 1. **Single line comment:** The comments that begin with // are single line comments. The Compiler simply ignores everything following // in the same line.
 2. **Multiline comment:** The multiline comment begin with /* and end with */ . This means everything that falls between /* and */ is consider a comment even though it is spread across many lines.

❖ Pre-processor Directives (Linker Section):

- The linker section begins with a hash (#) symbol. #include is a preprocessor directive.
- It is a signal for preprocessor which runs the compiler.
- The statement directs the compiler to include the header file from the C++ Standard library.
- **Example:**

```

#include<iostream.h>
#include<conio.h>
#include<math.h>

```

❖ Definition:

- In this section, we can define constants, expressions, structures, functions, classes and objects.
- After the linker section gets executed, the definition section is executed by the compiler and this section is optional.
- **Example:**

```

#define PI 3.14

```

❖ Global Declaration

- We can declare variables here and those variables which are declared before the main function or any other function then the life or scope of such variables remain throughout function and can be used by other functions of the program.

❖ main() Function:

- As the name itself indicates, this is the main function of every C++ program.
- Execution of a C++ program starts with main ().
- No C++ program is executed without the main () function.
- The function main () should be written in the lowercase letter and shouldn't be terminated with a semicolon.
- It calls other library functions and user-defined functions.
- There must be one and only main () in every C++ program.

❖ Braces { }:

```
{  
.....;  
.....;  
}
```

- The statements inside any function including the main () function is enclosed with the opening and the closing braces.

❖ Declarations:

- The declaration is the part of the C++ program where all the variables, arrays, and functions are declared with their basic data types.
- This helps the compiler to allocate the memory space inside the computer memory.

• Example:

```
int sum, x, y;
```

❖ Statements:

- These are instructions to the computer to perform some specific operations.
- These statements can be expressions, input-output functions, conditional statements, looping statements, function call and so on. They also include comments.
- Every statement end with semicolon “;” except control statements.
- Semicolon “;” also known as **Terminator**.

• Example:

```
cout<<”Welcome to Computer Science Class”;
```


Example: A Simple C++ program to display a message on the screen:

```
#include<iostream.h>
#include<conio.h>
void main ()
{
    clrscr();
    cout<< "Welcome to Computer Science Class";
    getch();
}
```

The program produces following output:

Welcome to Computer Science Class

➤ **Library Function:**

- C++ provides many built-in functions that save the programming time.
- They include mathematical functions, character functions, string functions, and console input-output functions.

➤ **Mathematical Function:**

- Some of the important mathematical functions in header file **math.h** are

Function	Meaning
sqrt(x)	Square Root of X
pow(x, y)	x raised to the power y
sin(x)	Sine of an angle x (measured in radians)
cos(x)	Cosine of an angle x (measured in radians)
tan(x)	Tangent of an angle x (measured in radians)
asin(x)	$\text{Sin}^{-1}(x)$ where x (measured in radians)
acos(x)	$\text{Cos}^{-1}(x)$ where x (measured in radians)
exp(x)	Exponential function of x (ex)
log(x)	Logarithm of x
log ₁₀ (x)	Logarithm of number x to base 10
abs(x)	Absolute value of integer number x
fabs(x)	Absolute value of real number x

➤ **Character Function:**

- Some of the important mathematical functions in header file **ctype.h** are

Function	Meaning
isalpha(c)	It returns True if C is an uppercase letter and false if c is lowercase.
isdigit(c)	It returns True if c is a digit (0 to 9) otherwise false
isalnum(c)	It returns True if c is digit from 0 through 9 or an alphabetic character (either uppercase or lowercase) otherwise false
islower(c)	It returns True if c is a lowercase letter otherwise False
isupper(c)	It returns True if c is a uppercase letter otherwise False
toupper(c)	It converts c to uppercase letter
tolower(c)	It converts c to lowercase letter.

➤ String Function:

- Some of the important mathematical functions in header file **string.h** are

Function	Meaning
strlen(s)	It gives the no. of characters including spaces present in a string s.
strcat(s1, s2)	It concatenates the string s2 onto the end of the string s1. The string s1 must have enough locations to hold s2.
strcpy(s1, s2)	It copies character string s2 to string s1. The s1 must have enough storage locations to hold s2.
strcmp((s1,s2)==0) strcmp((s1,s2)>0) strcmp((s1,s2)<0)	It compares s1 and s2 and find out whether s1 equal s2, greater than s2 or s1 less than s2.
strcmpi((s1,s2)==0) strcmpi((s1,s2)>0) strcmpi((s1,s2)<0)	It compares s1 and s2 ignoring case and find out whether s1 equal s2, greater than s2 or s1 less than s2.
strrev(s)	It converts a string into its reverse.
strupr(s)	It converts a string s into upper case.
strlwr(s)	It converts a string into lower case.

➤ Console I/O Function:

- Some of the important mathematical functions in header file **stdio.h** are

Function	Meaning
getchar()	It returns a single character from a standard input device (keyboard). It takes no parameter and the returned value is the input character.
putchar()	It takes one argument, which is the character to be sent to the output device.

	It also returns this character as a result.
gets()	It gets a string terminated by a newline character from the standard input stream stdin.
puts()	It takes a string which is to be sent to output device.

➤ General Purpose standard Library Function:

- Some of the important mathematical functions in header file **stdio.h** are

Function	Meaning
randomize()	It initialize/seeds the random number generator with a random number.
random(n)	It generates a random number between 0 to n-1.
atoi(s)	It converts string s into a numerical representation.
itoa(n)	It converts a number to a string.

➤ Some more Function:

- getch() and getche() functions
- The genral form of the getch() and getche() is

```
ch = getche( );
ch1 = getch( );
```
- ch and ch1 are the variables of type character. They take no argument and require the **conio.h** header file.
- On execution, the cursor blinks, the user must type a character and press enter key.
- The value of the character returned from getche() is assigned to ch.
- The getche() function echoes the character on the screen.
- Another function, getch(), is similar to getche() but does not echo character to the screen.

CHAPTER 7 – INTRODUCTION TO C++ BLUE PRINT

VSA (1 marks)	SA (2 marks)	LA (3 Marks)	Essay (5 Marks)	Total
01 Question	-	01 Question	01 Question	09 Marks
