Syllabus

Unit no.	Content	Lectures
1.	**Introduction to C++** 1.1. Basics of C++, 1.2. Structure of C++ Program, keywords in C++, 1.3. Data types hierarchy in C++, 1.4. Operators in C++: Scope resolution operator, Insertion and Extraction operator New and Delete operators. 1.5. Reference variable. 1.6. Manipulators function: endl, setw, set fill, set precision	6
2.	**Object oriented Concepts** 2.1. Object oriented concepts 2.2. Features, 2.3. Advantages and Applications of OOP 2.4. Difference between Procedure oriented programming and object oriented programming.	2
3.	**Classes and Objects** 3.1. Structure sand class, Class, Object, Access specifies, 3.2. Class members, 3.3. Defining member functions :Inside and outside the class definition, 3.4. Creating objects. String class, operation on string, Array of objects. 3.5. 'this' pointer.	10
4.	**Function in C++** 4.1. Call by reference, Return by reference, 4.2. Function overloading and default arguments 4.3. Inline function 4.4. Passing and returning objects from functions, Static class members 4.5. Friend Concept – Function, Class	6
5.	**Constructors and Destructors** 5.1. Memory allocation and static data members 5.2. Definition of constructor Types of constructors: Default Constructor 5.3. Constructor with default arguments 5.4. Parameterized Constructor 5.5. Copy Constructor 5.6. Overloaded constructors in a class 5.7. Destructors	4
6.	**Operator overloading** 6.1. Introduction, rules of operator overloading 6.2. Operator overloading: 6.3. Unary and binary operators, 6.4. Comparison, arithmetic assignment operator 6.5. Overloading new & delete operators 6.6. Overloading without friend function and using friend function,	6

7.	**Inheritance** 7.1. Introduction 7.2. Types of Inheritance: Single inheritance Multiple inheritance, Multilevel inheritance Hierarchical inheritance Hybrid inheritance. 7.3. Derived Class Constructor sand Destructors 7.4. Ambiguity in multiple Inheritances, virtual base classes, Abstract base class.	8
8.	**Virtual Function &Polymorphism** 8.1. Introduction, Pointer to object, Pointer to derived 8.2. Class, Overriding member functions, Virtual function, Rules for virtual functions, pure virtual function, Run- time type information (RTTI)	6
9.	**Working with files** 9.1. File operations – Text files, Binary files 9.2. File stream class and methods 9.3. File updation with random access 9.4. Overloading insertion and extraction operator	4
10.	**Templates** 10.1 Introduction function templates, function templates with multiple parameters 10.2 Overloading of template functions, Class Templates, class template with multiple Parameters, member function templates 10.3 Introduction to Standard Template Library(STL) 10.4 Components of STL 10.5 Containers 10.6 Algorithms 10.7 Iterators 10.8 Application of Container classes	5
11.	**Exception handling** 11.1 Exception Handling Mechanism 11.2 The try block 11.3 The catch block 11.4 Throw statement 11.5 The try/throw/catch sequence	3

*****

Contents

****

Introduction to C++

A Brief History of C++

e C++ programming language was designed and implemented by Bjarne Stroustrup at AT&T Bell boratories. It was a successor to the C language which was developed in 1972 by Dennis Ritchie Bell Labs. Stroustrup, as a part of his doctoral studies, studied various programming paradigms d languages. He was impressed with Simula67 which was an object oriented language developed 1967. Considering the efficiency and popularity of C, Stroustrup combined the features of nula67 and C into a new language and called it "C with Classes". It was renamed to C++ in 1983. e "++" indicates the evolutionary nature of the changes from C. *Figure 1.1* shows the history of ·+.

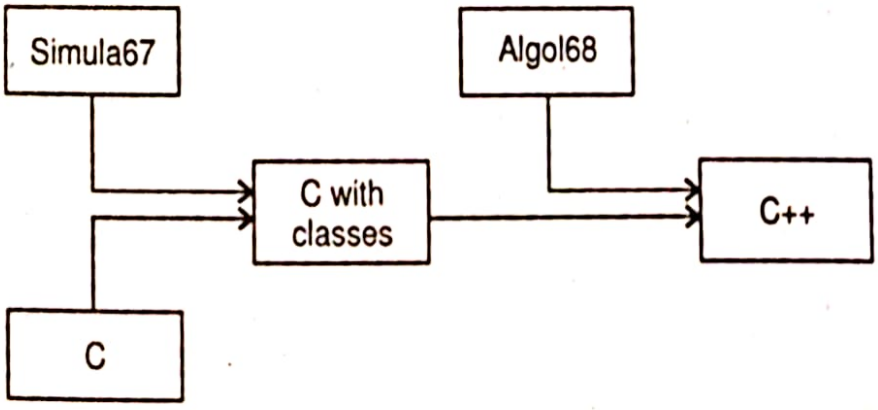

Figure 1.1: The History of C++

2. Differences between C and C++

There are several differences between the C language and C++ language. Some of these differences are highlighted here.

1. **Keywords:** C++ adds several new keywords. Some of these keywords are related to object oriented concepts. *Example*: public, private, protected, class etc.

2. **Operators:** C++ adds several new operators like scope resolution operator (::), memory operators - new and delete, typecasting operators - dynamic_cast, const_cast, static_cast, and reinterpret_cast.

3. **Functions:** There are many new types of functions in C++. These are inline functions, friend functions, static functions, overloaded functions, function overriding, functions with default parameters virtual and pure virtual functions.

4. **Boolean data type:** C++ has an additional data type called bool which represents either true or false values. C++ has two additional keywords 'true' and 'false'.

   C++ Example, bool answer = true;

5. **Object Oriented Differences:** C++ supports object oriented concepts such as inheritance, polymorphism, encapsulation etc, which C does not support.

6. **Flexible Declaration:** In C, all variable declarations within a scope occur at the beginning of that scope. In C++, you can declare variables anywhere within the program as long as the variables are declared before use.

7. **Struct, union and enum tag:** In C, when we create a structure, union or enumerated type, we have to use the keywords struct, union and enum respectively when declaring variables. In C++, the struct, union and enum tags are considered to be type names.

C example:

```
struct student
{
   int id;
   char name[20];
};
...
struct student s;
```

C++ example:

```
struct student
{
   int id;
   char name[20];
};
...
student s;
```

8. **Reference variables:** C++ supports special type of variable known as reference variables which are aliases to the original variables. This makes it easier to modify parameters in a function.

C example:

```
int a=10, b=20;
swap(&a, &b);
...
void swap(int *ptr_a, int
*ptr_b)
{
   //swap *ptr_a and *ptr_b
}
```

C++ example:

```
int a=10, b=20;
swap(a, b);
...
void swap(int& ref_a, int&
ref_b)
{
     //swap ref_a and ref_b
 }
```

9. **Comments:** C supports multiline comments while C++ supports single line as well as multiline comments.

C example:

```
/* This is a C comment
which can span several
   lines*/
```

C++ example:

```
//This is a single line comment
//from here to End of line
```

10. **Dynamic memory allocation:** In C, dynamic memory management is done using functions like malloc, calloc, realloc and free. In C++, dynamic memory allocation and de-allocation is done using operators (new and delete).

C example:

```
int *ptr=
malloc(5*sizeof(int));
. . .
free(ptr);
```

C++ example:

```
int *ptr = new int[5];
. . .
delete[] ptr;
```

11. **Input/Output:** In C, I/O operations are performed using functions like printf, scanf, etc. In C++, I/O operations are performed using predefined stream objects (cin, cout, fstream etc.) and operators (<< - insertion and >> - extraction).

12. **Namespaces:** C++ provides the concept of namespaces which allow grouping of a set of global classes, objects and/or functions under a name. All classes, functions and templates of the standard library are defined in the namespace *std*. Users can also create their own namespaces.

13. **Standard Template library:** C++ has a Standard Template Library (STL) which is a general-purpose C++ library and it is a rich collection of algorithms and data structures.

14. **Exception handling:** C++ provides keywords like try, catch and throw for exception handling.

15. **Template functions and classes:** C++ allows programmers to create template classes and functions without specifying what data types will be handled by the classes or functions. This is called generic programming.

16. **Run Time Type Inquiry:** Newer versions of C++ provide the facility of RTTI which allows us to identify the type of an object during the execution of the program.

3. Writing and Executing a C++ Program

There are the various stages involved in the process from creation to execution of a C++ program. They are as follows:

1. **Creating the Source Code:** A C++ program is written as ordinary text (called source). A small C++ program may only contain one file (called source file), but a larger program often contains several. The file containing the source code has to be a 'text' file with an extension such as .cpp.

2. **Compiling the Source Code:** The source file has to be converted to machine code using a compiler. The compiler takes .cpp or .cc files, preprocesses them and translates them into *object files* having extension .o or .obj.

   For compiling the source code from the operating system's command line, you should type the following statement.

   For example: On Linux, use the GNU C++ compiler gcc or g++ using the statement *gcc <filename>*.

3. **Linking the Object Code to create an Executable Code:** After successful compilation, the object file is given to the *linker*. This program combines the files, adds necessary libraries and creates an executable file with extension .exe.

4. **Executing the Program:** Once the executable file is created, you can run it by typing its name at the DOS command prompt or through the option provided by the compiler software. On Linux, you have to run the executable file a.out.

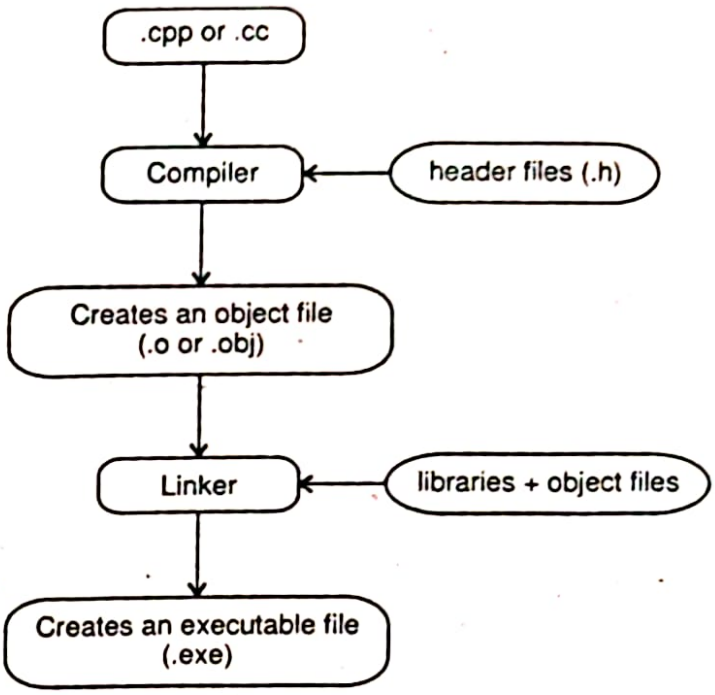

Figure 1.2: Compilation Steps

▶ Structure of a C++ Program

The typical structure of an object oriented C++ program is shown in the following table:

`inclusion of header files`
`definitions of global constants`
`declarations of global variables`

`class declaration`
`class member function definition`
`main function`

The first part of the C++ program is similar to a 'C' program. It contains the link section (#include), global constants (#define) and global variables.

The next section is the class declaration section where we create the classes to be used in the program. After the class declaration, the member functions of the classes are defined.

Finally, the main function is written where objects are created and operations are performed.

First C++ Program

Let us now see a simple C++ program which displays the message "Hello World !" on the screen. The program is given below.

Program : Hello World Program

```
Line 1    //This is my first C++ program
Line 2    //It displays a welcome message
Line 3    #include<iostream.h>
Line 4    using namespace std;
Line 5
Line 6    // main function
Line 7    int main()
Line 8    {
Line 9     cout<<"Hello World!"<<endl;
Line 10    return 0;
Line 11   }
```

If you compile and run this code, you will see the message "Hello World!" as output. Note that you don't have to type Line 1, Line 2, etc in the source code. It is given here for explanation purpose. *Let's take a look at the above program line by line.*

Line 1,2,6 : Single Line Comments: All the lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. They are used by the programmer to give additional information regarding the program. All characters following the // till the end of the line are treated as a part of the comment.

Line 3: Include directive: iostream.h is an object oriented library header file which includes the declarations of the standard input-output classes and objects which we use in the program. In this program, we include this because we are using cout which is a predefined output stream object.

Line 4: using namespace std: *std* is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope. *using namespace* are the new keywords of C++.

Line 7: int main(): This line corresponds to the beginning of the *main* function definition. The *main* function is the point from where all C++ programs begin their execution.

Line 8, 11: { }: These are the opening and closing braces of function main. All executable statements must be written within these braces.

Line 9: cout <<"Hello World !"<< endl; *cout* is the standard output stream in C++. It is used along with the insertion operator (<<). The string "Hello World !" is inserted into the output stream which is linked to the standard output device (monitor). *endl* is a manipulator which outputs a newline and then flushes the stream.

Line 10: return 0; : The main function is declared with a return type "int". The return statement causes the main() function to end and returns 0 which indicates that main has terminated normally.

▶ Simple C++ Program with Class and Object

C++ is an object oriented language. An object oriented program contains classes and objects. Let us now see a simple C++ program which contains a class called "Student". The program creates one student object and accepts and displays data for the student object.

```
//This program demonstrates classes and objects
#include<iostream.h>
using namespace std;
class Student
{
    char name[80];
    int id;
    public:
        void accept();
        void display();
};
void Student::accept()
{
    cout<<"Enter the name and id";
    cin>>name>>id;
}
void Student::display()
{
   cout<< "Name="<<name<<endl<<"ID ="<<id;
}
int main()
{
 Student s;
 s.accept();
 s.display();
 return 0;
}
```

In this program, we have created a class called "Student" which has two attributes: name and ic class has two member functions: accept() and display(). The functions are defined outside the using the scope resolution operator(::). In the main function, we create an object of the Student and perform accept and display operations on this object.

4. C++ Keywords

Keywords are *reserved words* and are predefined by the language. All the keywords shou written in lower case letters.

▶The keywords for the ANSI C language are

auto	double	int	struct	break	else	long	switch
case	enum	register	typedef	char	extern	return	union
const	float	short	unsigned	continue	for	signed	void
default	goto	sizeof	volatile	do	if	static	while

The following keywords have been added in C++

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

Operators: const_cast, dynamic_cast, reinterpret_cast, static_cast, new, delete

Exception handling: try, catch, throw

Access specifiers: public, private, protected

Object Oriented: class, friend, inline, operator, this, virtual, namespace, mutable

Generic programming: template

Constants: true, false

Run time type Inquiry: typeid, typename

Types: bool, wchar_t

Others: asm, explicit

5. Data Types in C++

▶ Fundamental Types

C++ provides the following fundamental built-in types: boolean, character, integer and floating-point. It also enables us to create our own user-defined types using enumerations and classes.

Name	Bytes	Description	Range
char	1	character or integer 8 bits length.	*signed:* -128 to 127 *unsigned:* 0 to 255
int	2 or 4	Integer. Its length depends on the length of the system's word type, thus in MSDOS it is 16 bits long, whereas in 32 bit systems (like Windows 9x/2000/NT and systems that work under protected mode in x86 systems) it is 32 bits long (4 bytes).	*signed*: -32768 to 32767 *unsigned*: 0 to 65535
short	2	Type short int (or simply short) is an integral type that is larger than or equal to the size of type char, and shorter than or equal to the size of type int.	*signed*: -32768 to 32767 *unsigned*: 0 to 65535
long	4	Type long (or long int) is an integral type that is larger than or equal to the size of type int. Objects of type long can be declared as signed long or unsigned long. Signed long is same as long.	*signed*: –2147483648 to 2147483647 *unsigned*: 0 to 4294967295
float	4	floating point number.	3.4e - 38 to 3.4e - 38 (7 digits)
double	8	double precision floating point number.	1.7e - 308 to 1.7e + 308 (15 digits)
long double	10	long double precision floating point number.	3.4e - 4932 to 1.1e + 4932 (19 digits)
bool	1	Boolean value. It can take one of two values: true or false	true or false
wchar_t	2	Wide character. It is designed as a type to store international characters of a two-byte character set.	wide characters

Note Signed means the number can be positive or negative. Unsigned means the number must be positive.

▶ User Defined Data Types

C++ allows user defined types to be created. Apart from enumerated type, it supports creating user types or classes using keywords struct and class.

1. **Structure:** We have used structures and unions in C. They are also supported by C++. However, the structure or the union tag becomes a new typename in C++. In the following *example*, "student" becomes a new type name and we can use it to create variables just like any other type.

```
structure tag
{
   type element1;
   type element2;
   .
   .
};
```

For example,

```
struct student
{
   int rno;
   char name[20];
   float perc;
};
```

To create variables of this type, we can use student as a type. There is no need of using keyword struct again.

For example,

```
student s1, arr[10];
```

2. **Unions:** Unions allow a portion of memory to be shared by different variables. Only variable can be accessed at a time. Its declaration and use is similar to the one of the structu but its functionality is totally different:

```
union tag
{
   type element1;
   type element2;
   .
   .
}object_name;
```

All the elements of the *union* declaration occupy the same space of memory. Its size is tha the largest element of the declaration.

For example,

```
union mytypes
{
   char c;
   int i;
   float f;
};
```

The size of this union will be that of the size of a float type. To create a variable, we can u the union tag like a typename.

For example,

```
mytypes t1;
```

Here, t1 can either store a char (t1.c) or int (t1.i) or a float (t1.f).

3. **Class:** A class is one of the most important concepts in an Object Oriented system. A class is a user defined type which has data members as well as member functions. In C++, a class can be created using the keywords *struct* as well as *class*. *For example*, a class account that has data members and operations:

```
class account
{
     public:
          void open();
          void close();
          void withdraw(int amount);
          void deposit(int amount);
     private:
          int ac_number;
          char name[80];
          float balance;
};
```

Here, account becomes a new type. To create objects of this type, we can use the class name followed by the object name as shown below.

For example,

```
account acc1, acc[10];
```

6. Input and Output (cin and cout)

In C / C++ all standard input and output operations are performed using streams. A stream is a sequence of bytes. The sequence of bytes flowing into a program is called as *input stream* and the one flowing out from the program is called as the *output stream*. In other words, a program extracts the bytes from an input stream and inserts bytes into an output stream.

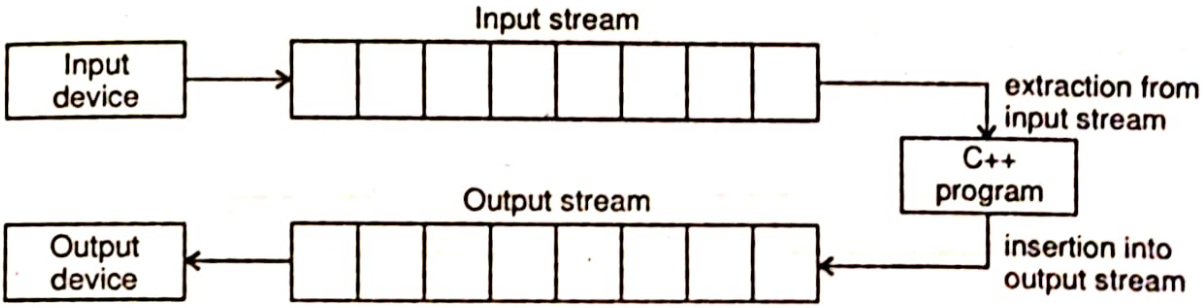

Figure 1.3: Input and Output streams

The C++ I/O system contains a hierarchy of stream classes. C++ contains several pre-defined objects of stream classes. These include *cin* and *cout* for console input and output operations.

6.1 Output Stream(cout)

For displaying data on the standard output device, the **cout** stream object is used. It is a prede object of the 'ostream' class. It is used with insertion operator "<<" which inserts data int output stream.

Syntax:

```
cout << data;
```

The << operator is known as *insertion operator* since it inserts the data that follows it into the st that precedes it. The data to be inserted into the output stream can be a variable name, constant expression.

Examples

```
1.  cout << "Hello";
2.  cout << a+b;
3.  cout << fact(n); //fact() returns a value
4.  cout << 250;
```

The following diagram shows how cout works.

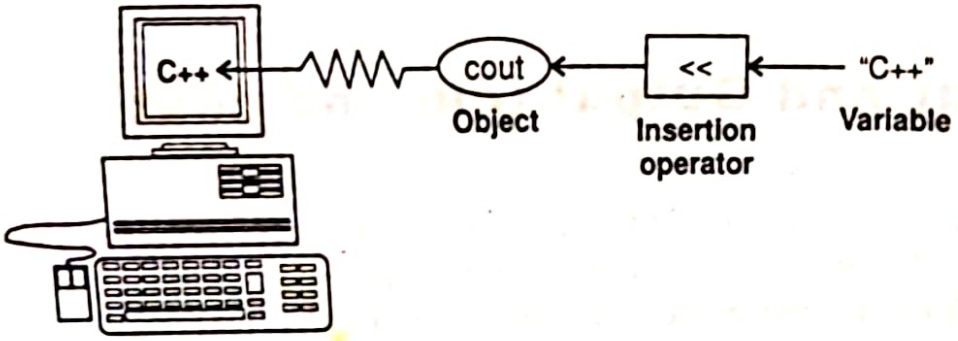

Figure 1.4: Output using Insertion operator

The *insertion* operator (<<) may be used more than once in the same instruction to display mult data values. This is called *cascading* of operators.

Syntax:

```
cout << data1 << data2 << data3 << ...;
```

Examples

```
1.  cout << " The sum of "<< a << " and" << b << "  = "  << a+b;
2.  cout << " I am "  << age << " years old";
3.  cout << x << y << z;
```

Cascading is useful when we want to display many data items without writing multiple cout statements.

It is important to notice that *cout* does not add a line break after its output unless we explicitly indicate it, therefore, the following sentences:

```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

will be shown on screen as followos:

```
This is a sentence. This is another sentence.
```

In order to perform a line break on output we must explicitly order it by inserting a new-line character \n or use the manipulator *endl*.

```
cout << "First sentence\n";
cout << "Second sentence" << endl << " Third sentence";
```

6.2 Input Stream (cin)

The standard input stream object is 'cin' which is used to read data from a standard input device like keyboard. It is a predefined object of the class 'istream'. It is used with the extraction operator ">>". This must be followed by the variable name that will store the value being read from the stream.

Syntax:

```
cin >> variable ;
```

The following diagram shows how a value is read using cin.

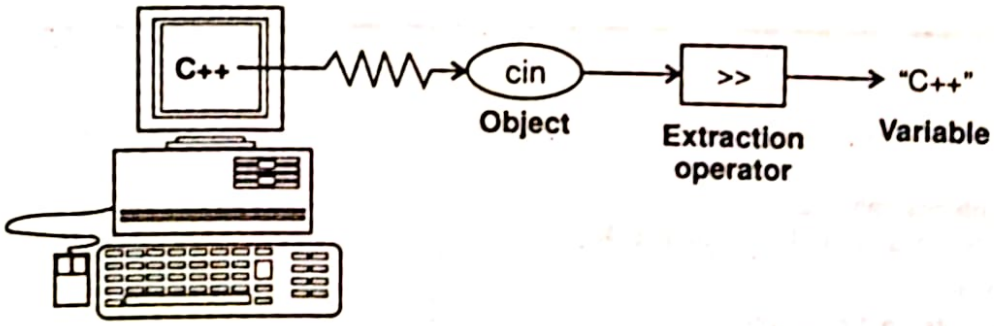

Figure 1.5: Input using extraction operator

For example:

```
int age;
cin >> age;
```

cin only reads the input after the RETURN key has been pressed. You can also use *cin* to read than one data value from the user by cascading the extraction operator.

Syntax:

```
cin >> variable1 >> variable2 >> ... ;
```

For example,

```
cin >> a >> b;
```

is equivalent to: `cin >> a;  cin >> b;`

The following program shows how cin and cout can be used to read and display values of diff types.

Program: Using cin and cout

```
#include<iostream.h>
int main()
{
  int i ;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << endl;;
  char ch;
  cout << " Enter a character : ";
  cin >> ch;
  cout <<"You entered : " << ch << " and its next character = " <<ch+1;
  return 0;
}
```

Output

```
Please enter an integer value: 20
The value you entered is 20 and its double is 400
Enter a character: a
You entered: a and its next character = b
```

7. New Operators in C++

All C language operators are valid in C++. In addition, C++ introduces some new operators which are given below.

Operators	Meaning
::	Scope Resolution operator
:: *	Pointer - to member declaration
. *	Pointer - to member operator
→ *	Pointer- to member operator
new	Memory allocation operator
delete	For releasing memory

As seen in the table above, there are several operators which have been introduced in C++. We shall study each of these with *examples* in the following sections.

7.1 Scope Resolution Operator (::)

The Scope Resolution operator (::) is a very important operator in C++. It can be used as a unary or as a binary operator. There are two uses of this operator:

i. To unhide the global variable that might have got hidden by the local variables.

ii. To define a member function outside the class.

iii. To access class variables (static members and functions).

The first use of this operator is to allow access to the global variable even though there is a local variable of the same name within a function. The use of this operator in front of the variable name instructs the system to use the global variable rather than the local variable.

Syntax:

```
::variable
```

Consider the *example* illustrating the use of scope resolution operator.

Program: Scope resolution operator

```
#include<iostream.h>
int x = 10; //global variable
int main()
```

```
{
   int x = 20;
   cout<<"Local x = " << x <<endl;
   cout<<"Global x = "<< ::x <<endl;
   {
     int x = 30;
     cout<<"In nested block, x = "<< x <<endl;
     cout<<"Global x = "<< ::x <<endl;
   }
return 0;
}
```

Output

```
Local x = 20
Global x = 10
In nested block, x = 30
Global x = 10
```

The second use of this operator is to define member functions of a class outside the class. When member function is defined outside the class, we have to indicate that it belongs to the class. This done by using classname followed by scope resolution operator.

Syntax:

```
returntype class-name :: function-name(arguments)
{
//body
}
```

Consider the following *example*:

Program : Scope resolution operator for member function

```
#include<iostream.h>
class A
{
  void display();
};
void A::display()
{
  cout<<"In display function of class A";
}
int main()
{
   A obj ;
  obj.display() ;
  return 0;
}
```

The third use of the scope resolution operator is to access static data members and static functions o a class. Class members which are declared static do not belong to an object but belong to the class In such a case, the member is accessed using ::

Syntax:

```
class-name :: static-member-name
```

In the following *example*, we declare a data member x which belongs to the class.

Program : Scope resolution operator with static member

```
#include<iostream.h>
class A
 {
  public:
   static int x;        //static class member
 };
 int A::x = 10;         //initialization
 int main()
 {
   cout<<"Class x = "<< A::x <<endl;
   return 0;
 }
```

Output
Local x = 20
Class x = 10

7.2 Member Dereferencing Operators

In C++, a class can contain data members as well as member functions. The data members can be of different types including pointers. Moreover, a class member may also be accessed using a pointer. For this purpose, C++ provides three operators to access members using pointers and access data members which are pointers.

i. ::*

This operator is used to declare a pointer to a member of a class. *For example*, if x is a data member of class A, a pointer to x can be declared as follows:

```
int A::*ptr = &A::x;
```

ii. .*

This operator is used to access a member using object name and a pointer to that member. *For example*, in the above code, we had declared a pointer to a class member x. To access that member using the pointer, we can use the .* operator as shown.

```
A aobj;           //object of class A
aobj.*ptr = 20;   //use pointer to change value of x
```

iii. →*

This operator is used to access a member using a pointer to the object and a pointer to that member. In the above *example*, we have used a class object to access the member using the pointer. Similarly, we can also use a class pointer to access the member using the pointer.

```
A *a_ptr;           //pointer to class A
a_ptr = new A();    //create object dynamically
a_ptr->*ptr = 20;   //use class pointer to access pointer of x
```

The following program illustrates the use of these three operators.

Program : Scope resolution operator with class

```
#include<iostream.h>
class A
{
  public:
 int x;
};
int main()
{
   int A::*ptr = &A::x;       //declare pointer to member
   A obj , *a_ptr;
   obj.*ptr = 20;             //access using .*
   cout<<"Using .* : x = "<< obj.*ptr <<endl;
   a_ptr = new A();
   a_ptr->*ptr = 30;          //access using ->*
   cout<<"Using ->* : x = "<< a_ptr->*ptr <<endl;
   return 0;
}
```

Output

```
Using .* : x = 20
Using ->*: x = 30
```

7.3 Memory Management Operators (new and delete)

In C, dynamic memory allocation and deallocation is done using functions (malloc, calloc, realloc and free). C++ provides two unary operators – *new* and *delete* to allocate and free memory dynamically. Memory is allocated using new and released using the delete operator. They are also called *free store operators* because they operate on the dynamic memory called the free store.

▶ "new" Operator

The new operator is used to allocate memory for a data element of any type (built-in or user defined).

▶ "delete" Operator

When the memory allocated dynamically is no longer needed, it should be freed so that it becomes available for future requests. This can be done by using the delete operator.

These operators will be covered in detail in later chapters.

8. Reference Variables

A reference variable is a new name given to an existing variable. It is an alias of the original variable. Since it is a reference to another variable, the reference variable cannot exist independently i.e. it can only refer to an *existing* variable and it does not occupy any additional memory.

The syntax of creating a reference variable is:

```
dataType& referenceName = variableName;
```

Examples:

1.
```
int x = 10;        // x is an integer variable
// rx is a reference variable
int& rx = x;
```

2.
```
float f = 4.78;    // f is a float variable
float& rf = f;     // rf is a reference variable
```

Note The variables x and f must exist in the above *examples*. A reference variable has to be initialized. We cannot declare a reference as:

```
int &rx;    //invalid
```

When we create a reference variable, we are not creating a new variable i.e. no memory is allocated to the reference variable. The original name and the reference name both refer to the same memory location. The value can be accessed using both, the original variable name as well as the reference name. The value can be changed using either the original name or the reference name.

To access the reference, do not use the ampersand operator; just the name of the reference is sufficient. The following *example* illustrates this.

Example

```
int x = 10;
int& rx = x;
cout<<"x = "<<x <<endl;
cout<<"rx = "<<rx <<endl;
```

Output

```
x = 10
rx = 10
```

Any changes made to rx or x will be made to the same variable.

```
x = 20;          //modify using original name
cout<<"x = " <<x <<endl;
cout<<"rx = "<<rx <<endl;

rx = 30 ;        //modify using reference
cout<<"x = "<<x <<endl;
cout<<"rx = "<<rx <<endl;
```

Output

```
x = 20
rx = 20
x = 30
rx = 30
```

One important use of reference variables is to pass parameters to functions. If we use reference variables in a function, we can access the variables of the calling function directly without using pointers. Any changes made to the reference variable will be made to the original variable.

The *following example* illustrates this.

Example:

```
void swap(int& a, int& b)
{
   int temp ;
   temp=a ; a=b ; b=temp ;
}
int main()
{
   int x = 10, y=20;
   cout<<"Original x = "<<x <<" y = "<<y endl;
   swap(x,y) ;
   cout<<"Swapped x = "<<x <<" y = "<<y endl;
}
```

Output

```
Original x = 10 y = 20
Swapped x = 20 y = 10
```

In the above *example*, a and b are reference variables for x and y respectively. In the function, a and b are swapped. Hence, actually x and y are swapped.

9. Manipulators

Manipulators are special functions used on stream objects for performing formatting operation on the stream. C++ provides various stream manipulators that perform formatting tasks to alter the state of the stream. The following table lists some important manipulators used in C++.

Function	Purpose
endl	Insert newline character into the stream and flushes the buffer.
setw(int)	Sets the width of a field to the specified number of characters.
setfill(char)	Sets the fill character used to pad fields.
setprecision(int)	Sets the precision of floating-point values to the specified number of digits.

1. **endl:** This manipulator is similar to the '\n' character i.e. it inserts a newline character into the stream. Additionally, it also clears the output buffer.

Syntax:

```
endl
```

Example,

```
cout <<"Hello"<<endl<< "World";
```

Output

Hello
World

2. **setw() :** The setw() manipulator is used to set the current width for output operation. Width is defined as the minimum number of characters to display with each output.

Syntax:

```
setw(int w)
```

If the number of characters to be displayed are less than the field width, the extra field will be padded with the current fill character (space by default). If the size of the value exceeds the minimum field width, the width setting will be ignored. No values are truncated.

Note The setw() setting is applicable only to one data item which is displayed immediately afterwards. After that, the width setting reverts back to the default.

Example,

1.
```
cout << setw(5)<<"2";
```

Output

				2

2.
```
cout<<setw(5);
cout << 123.456;
cout <<"ABC";
```

Output

1	2	3	.	4	5	6	A	B	C

3. **setprecision():** The setprecision() manipulator is used to set the precision of the displa floating point numbers i.e. number of digits to be displayed after the decimal point. default precision is 6.

Syntax:

```
setprecision(int p)
```

Example,

```
float f1 = 15.16123, f2 = 25.12467;
cout <<setprecision(3);
cout <<f1 <<"\n" <<f2;
```

Output

15.161
25.125

Note: Unlike setw(), setprecision() setting remains until it is reset.

4. **setfill():** The setfill() manipulator sets the current fill character. The fill character is defined the character that is used for padding when the number of characters to be displayed is smal than the specified width. The default fill character is the space character.

Syntax:

```
setfill(char ch)
```

For example,

```
cout<<setw(10)<<setfill('*');
cout << "Hello";
```

Output

*	*	*	*	*	H	e	l	l	o

Note: The setfill() setting remains until changed.

Solved Example

1. What will be the output of the following statement? (Assume there are no syntax errors

```
cout<<setw(10)<<setprecision(2)<<setfill('*')<<123.1234<<"Rs";
```

Output

*	*	*	*	1	2	3	.	1	2	R	s

Explanation

cout<<setw(10); → sets the display width to 10.

cout<<setprecision(2) → sets the floating point precision to 2.

cout<<setfill('*') → fills the unused spaces by '*'

The width setting is only applied to the value 123.1234. Only 2 precision digits are displayed and the extra spaces are filled by '*'.

EXERCISES

A. Short answer Questions

1. List the additional operators in C++.
2. State the purpose of new and delete.
3. State the purpose of scope resolution operator.
4. What are cin and cout?
5. Define reference variable.
6. "A reference variable must be initialized". State True/False.
7. Give the syntax of creating a reference variable.
8. Define manipulator.
9. List the important manipulators in C++.

B. Long answer Questions

1. What are the features of the C++ language?
2. How can a comment be written in a C++ program?
3. Write the differences between C and C++.
4. Explain the structure of a C++ program.
5. Write a short note on scope resolution operator.
6. Explain the use of new and delete with examples.

7. Write the difference between dynamic memory management in C and C++.

8. Write a short note on reference variables.

9. What is a manipulator? Explain the setw and setfill manipulators.

C. Programming Exercises

1. Write a C++ program that prints the following message on screen "My first C++ prog (Use cout statement).

2. Find errors if any in the following C++ statements

   a. cout >> "X="x;
   b. cin <<x;<<y;
   c. cout <<\n"Name:" << name
   d. cout << "Enter value:"

3. Write the definition for a class called Rectangle that has floating point data members l and width. The class has the following member functions:

   void setlength(float) to set the length of data member

   void setwidth(float) to set the width of data member

   float perimeter() to calculate and return the perimeter of the rectangle

   float area() to calculate and return the area of the rectangle

   void show() to display the length and width of the rectangle

   Write main function to create two rectangle objects and display each rectangle and its area perimeter.

Unit 2

Object Oriented Concepts

1. Overview of Procedure Oriented Programming

In Procedure oriented programming, a program is considered as a "sequence of tasks to be done". Hence, the emphasis is on the procedure or algorithm. To manage the complexity of the task, usually a modular approach is used. As a result, not enough attention is given to data. If the same data is needed by many fun ctions, programmers tend to use global data. This affects the security of data. Such programs use the **top-down** approach. Programs written using most high level languages such as COBOL, FORTRAN and C use the **procedure-oriented** approach.

A typical program structure for procedural programming is shown in the following *figure*.