BCA - 2nd Year

3rd semester

Object Oriented Programming Using C++

UNIT -1

* **C++ :-** C++ is an object oriented programming language. It was developed at AT & T lab, bell laboratories in early 1979. It was rename as C++. C++ language is an extension to C programming language and supports classes, inheritance, functions, loading and operator overloading which where not supported by C programming language.

The object oriented features in C++ is helpful in large program with extensibility and easy to maintain the software after sale to costumer. It is helpful to make the real world problems properly.

C++ is a middle level programming language developed by Bjarne Stroustrup starting in 1979 at Bell labs. C++ runs on a variety of platforms such as Windows, MAC OS and a various versions of UNIX.

Object oriented programming including the four pillers of object oriented development -

1- Encapsulation
2- Data hiding
3- Inheritance
4- Polymorphism

* **Learning C++ :-** The most important thing while learning C++ is to focus on concepts. The purpose of learning a programming language is to become more effective at designing and implementing new systems and at maintaining old ones.

* **Characters used in C++ :-**

1- Alphabats
2- Uppercase letters A to Z.
3- Lowercase letters a to z.
4- Numbers 0 to 9
5- Special characters -

+	Plus	"	Double quotes
-	Minus	&	Ampersand
*	Asterisk	#	Hash
^	Caret	$	Dollar
/	Slash	%	Percent
|	Vertical bar	>	Greater than
~	Tilde	<	Less than
)	Close parenthesis	=	Equal to
(	Open parenthesis	!	Exclamation mark

White space character - A character that is used to produce blank space when printed in C++ is called white

space character. These are space tags, new lines and comments.

* 60 Keywords or reserve words :- Keywords are reserved words. C++ language use the following keywords which are not available to users to use them as variables, function names. Generally all keywords are in lower case although uppercase of same name can be use as an identifier.

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

Add by ANSI C++ new keywords –

bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar

Keywords are predefine reserve words used in programming that have special meaning to the compiler. Keywords are the part of the syntax and they cannot be used as an identifier.

For example – Keyword

int house → variable

Here int is a keyword that indicates house is a variable identifier to type integer.

Keyword cannot be changed by the user. They are always written in lower case.

* **Identifiers :-** Identifiers are user-defined words. Identifiers refers to name given to entities such as variables, fuctions, arrays etc.

Some rules of naming identifiers are –

1- First character should be an alphabat or underscore.
2- The name should not be a keyword.
3- Since C++ is case sensitive the uppercase and lowercase characters are consider different.
~~4~~- For ex – Code, code, CODE
4- A valid identifier can have character both upper and lower case, digits and

underscore.

Some examples of valid identifiers are - Nirbhay, Tech_fully, Youtube123.

Some examples of invalid identifier names are -

(i) 1234Youtube → first character should be an alphabat
(ii) rec# → # is a special character.
(iii) account no → blank space is not allowed.

* **Variable** :- A variable is a name of memory location. It is used to store data. It's value can be changed and it can be used many times. It is a way to represent memory location through symbol, so that it can be easily identify.

For ex -
```
int x;
float y;
char z;
```

* **Data types** :- A data type specifies the type of data that a variable can store such as - integer, floating, characters etc. There are 4 types to data types in

C++ language.

	Types	data types
1.	Basic data types	int, char, float, double etc
2.	Derived data types	array, pointer etc
3.	Enumeration data type	enum
4.	User defined data type	structure

The basic data types are integer based and floating point based. C++ language supports both signed and unsigned literals. The memory size of basic data types may change according to 32 or 64 bit operating system.
Let us see the basic data types, its size is given according to 32 bit OS.

Data type	Size	Range
char	1 byte	-128 to 127
signed char	1 byte	128 to 127
unsigned char	1 byte	0 to 127
short	2 bytes	-32768 to 32767
signed short	2 bytes	-32768 to 32767
unsigned short	2 bytes	0 to 32767
int	2 bytes	-32768 to 32767
signed int	2 bytes	-32768 to 32767
unsigned int	2 bytes	0 to 32767
short int	2 bytes	-32768 to 32767
signed short int	2 bytes	-32768 to 32767

unsigned short int	2 bytes	0 to 32767
long int	4 bytes	
signed long int	4 bytes	
unsigned long int	4 bytes	
float	4 bytes	
double	8 bytes	
long double	10 bytes	

* Semi colon and blocks in C++ :- The semi colon is a statement terminater that is each individual statement must be ended with a semi colon. It indicates the end of one logical entity.
For ex- following are there different statements -

```
x = y;
y = y + 1;
```

A block is a set of logical connected statement that are surrounding by openning and closing braces.
For ex-

```
{
cout << "Hello World";
return 0;
}
```

* **Comments in C++ :-** Program comments are explainatory statement that you can include in the C++ codes that you write and have any one reading its source code. All programming languages allow for some form of comment. C++ supports single-line and multi-line comments. All characters available inside any comments are ignore by C++ compiler.

For ex-

```
{
/* cout << "Hello World"; */
   Cout << "Hello";
   return 0;
}
```

When the above code is compiled it will ignore cout << "Hello World"; and final executable will produce the following result - Hello

* **Variable Definition :-** A variable definition mean that the programmer write some instruction to tell the compiler to create the storage in a memory location.

Syntax -

```
data_type variable_name, variable_name;
```

Here data-type means the valid C++ data type which include int, float, char,

double, boolean and variable list is the list of variable names to be declared which is seperated by comma.
For ex -

```
char letter;  → Variable definition
float area;
```

* **Variable initialization in C++:-** Variables are declared in the above example but none of them has been assign any value. Variable can be initialized and the initial value can be assign along with their declaration.
Syntax -

```
data_type variable name = value;
```

For example -

```
char letter = 'A';
float area = 26.5;
```

* **Scope of variables:-** All the variables have their area of functioning and out of that boundry they do not hold their value, this boundry is called scope of variables. For most of the cases its between the curly braces in which variables is declared that a variable exist not outside it. We can

divide variables in two main types-

1- Global variables

2- Local variables

Program 1- A program to understand the use of cout object.

Ex-

```
#include <iostream.h>
#include <conio.h>
void main() {
clrscr();
int length;
length = 10;
cout << "The length is";
cout << length;
getch();
}
```

→ Insertion operator

Output :- The length is 10.

Program 2:-

```
#include <iostream.h>
#include <conio.h>
int a = 10; → global variable
int main() {
int a = 15; → local variable
cout << "local a": << a << "global a:" << a;
}
```

Output - local a: 15
global a: 10

★ Cin (Console Input) :- It is similar to scanf() in C programming. It is used to read input value of variables from

the keyboard. It is an object of istream class.

Syntax –

cin >> variable_name;
(>> → Extraction)

Program3- A program to view the use of cout & cin objects.

```
#include <iostream.h>
#include <conio.h>
void main(){
    int a;
    cout<<"Enter any number:";
    cin >> a;
    cout<<" You are the best student:"<<a;
    getch();
    return 0;
}
```

Output :- Enter any number : 6
You are the best student : 6

* Cout :- It is similar to printf() in C programming. It is used to print / display value of variables. It is an object of ostream class.

Program4- A program to view the use of cout object-

```
#include <iostream.h>
#include <conio.h>
void main(){
    cout<<"Welcome to C++ programming\n";
    cout<<"Welcome\n BCA\n students";
    getch();
}
```

```
return 0;
}
```

Output :- Welcome to C++ programming
Welcome
BCA
students.

★ Abstraction :- Abstract classes are the way to achieve abstraction in C++. Abstraction in C++ is the process to hide the internal details, showing the functionality only. Abstraction can be achieve by two ways -

1. Abstract class
2. Interface

Abstract class and interface both can have abstract method which are necessary for abstraction. Data abstraction refers to providing only essential information to the outside world and hiding their background details to represent the needed information in program without presenting the details.

Data abstraction is a programming (and design) technique that relies on the seperation of interface and implementation. One real life example of a T.V. which you can turn on and off, change the channel, adjust the volume and add

external components such as speakers, DVD player but you do not know its internal details.

* **Encapsulation** :- Encapsulation is the process of enclosing within class and object the attributes and the methods. In other words, encapsulation allow wrapping upto data and functions into a single unit.

Advantages of Encapsulation-

1. Data hiding - Abstraction of external access a feature of a class.

2. Information hiding - The implementation of data are not known the users.

3. Implementation independency - A change in the implementation is done easily without affecting the interface.

* **Polymorphism in C++** :- The word polymorphism means having many forms. We can say polymorphism as the ability of a message to be display in more

than one form.
A real life example of polymorphism - a person at a same time can have different characteristics. Like a man at a same time is a father, a husband, an employee so a same person posses have different behaviour in different situation, this is called polymorphism. Polymorphism mainly divided in two types -

1- Compile time polymorphism (Compile time binding, early binding, static binding)
2- Run time polymorphism (Run time binding, late binding, dynamic binding)

1- Compile time polymorphism - This type of polymorphism is achieve by function overloading or operator overloading.

Function overloading - Where there are multiple function with same name but different parameters, then this function are said to be overloaded. Function can be overloaded by change in number of argument and change in type of argument.

2- Run time polymorphism - This type of polymorphism is achieve by function overriding. A function overriding on the

other hand access when a derived class has a definition for one of the member function of the base class that base function is said to be overriding.

* Inheritance :- The capability of a class to derive properties and characteristics from other class is called inheritance. Inheritance is one of the most important feature of object oriented programming.

Super class (Base class or parent class) - The class whose properties are inherited in sub class is called base or super class.

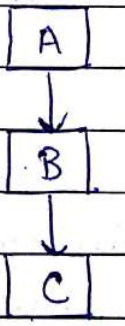
Sub class (Child class or Derived class) - The class that inherit properties from other class is called sub-class or derived class or child class.

Types of Inheritance – There are five Types of inheritance.

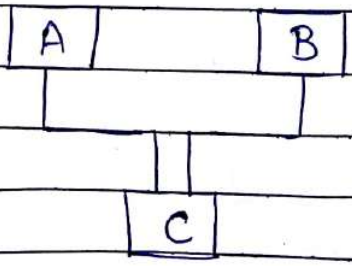
1. Single inheritance
2. Multiple inheritance
3. Multi-level inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

1- Single Inheritance –

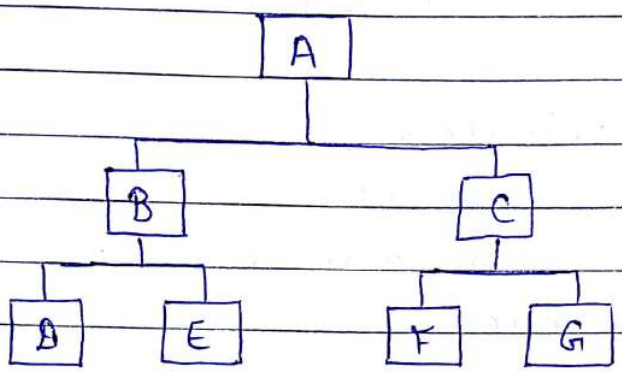
2- Multi-level Inheritance –



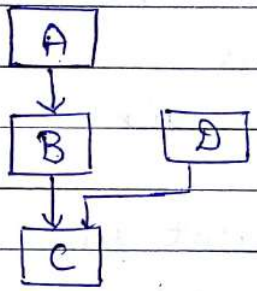
3- Multiple Inheritance –



4- Hierar



5- Hybrid Inheritance –



* Object – It is a class type variable.

* Class – A class is a group of common behaviour and functions. A class is a user-defined data-type like any other built-in data-type.

Example–

class → class student
{
- - - - - -
- - - - -
}

student stu1, stu2, stu3;
(stu1, stu2, stu3 → Objects)

* Example of Encapsulation, Inheritance —

```
class student
{
private:
    int id;
    char name;
public:
    void getdata();
    void display()
{
cout << id << \n << name. << end;
} };
int main()
{
student stu1, stu2, stu3;
-- --;
- - - -;
```

Access control specifies	Accessible to own class members	Object of class
Private	Yes	NO
Protected	Yes	NO
Public	Yes	Yes

★ Constructor :- It is the member function of the class called automatically when one object is created of that class. Constructor has the same name as that of class name and does not return any type. Also the constructor is always public. If we do not specify a constructor C++ compiler generate a default constructor for as accept no parameters and has an empty body.

(10) Program to call member functions using object of that class-

```
class comment
{
public:
void show()
{
cout <<" I am JAVA learner.";
```

```
}
};
int main()
{
comment yes;
yes.show();
getch();
return 0;
}
```

Output :-
I am Java learner.

(ii) 2. Example of constructor -

```
class comment
{
public:
comment() → function member constructor
{
cout<<" I love programming";
}
void show()
{
cout <<" I am Java learner";
}
};
int main()
{
comment yes;
yes.show();
getch();
return 0;
}
```

Output :-
I love programming.
I am Java learner.

There are three types of constructor -

1- Defaiult constructor

2- Parameterised constructor

3- Copy constructor

1- **Default constructor** — A constructor without any argument with default value for every argument is said to be default constructor.

What is significance of default constructor? Will the code be generated for every default constructor? Will there be any code inserted by computer to the user implemented default constructor behind the scenes.

The compiler will implicitly declare default constructor. If not provided by programmer will define it when in need. Compiler define default constructor is required to do certain initialization of class intervals. It will not touch the data members or plain old data type. Consider a class derived from another class will default constructor or a class containing another class object with default constructor. The compiler need to insert code to call the default constructor of base class/ object.

Example of default constructor —

Program 9.

```
class student → class name
{
private :
int sid ;
float marks ;   } data members declaration
public :
 student( ); → function declaration
 void showdetails( ) → simple function definition
 {
 cout <<"Student ID :" << sid <<"Marks" <<
                                    marks;
 } };
 student :: student( ) → constructor function definition
 {
 sid = 2;
 marks = 80.0;   } Data member value initialization
 }
 int main()
 {
 clrscr();
 student s1; → class object
 s1.showdetails(); → function calling
 getch();
 return 0;
 }
```

Output :- Student ID : 2 Marks : 80

2- **Parameterized constructor** — A default constructor does not have any parameter. But if you need a constructor can have parameter, this helps you to assign initial value to an object at the time of its creation. These are the constructors with parameter. Using this constructor you can provide different value to data members of different objects by passing the appropriate values as an argument.

(12) **Example of parameterized constructor —**

```
#include <iostream.h>
#include <conio.h>
class student
{
private:
int sid;
float marks;
public:
student (int i, float j)
{
sid = i;
marks = j;
}
void showdetails()
{
cout<<"student ID:"<<sid<<endl<<"marks"
<<marks;
```

```
};
void main()
{
student s1(3,700);
s1.showdetails();
getch();
}
```

student ID: 3
marks : 700

3- **Copy constructor** — It is a constructor of the form class name (class name &). Compiler will use the copy constructor whenever you initialize an instance using values of another instance of same type. A copy constructor takes a reference to an object of the same class as an argument.

Example-

(3)

```
# include <iostream.h>
# include <conio.h>
class student
{
int roll;
int marks;
```

```
public :
student (int m, int n) → function definition
{
roll = m;
marks = n;
}
student (student &t);
void showdetails()
{
cout << "\n Roll no:" << roll;
cout << "\n marks:" << marks;
}};
student :: student (student &t)
{
roll = t.roll;
marks = t.marks;
}
int main()
{
cout << "\n Parameterized constructor output \n";
student x(60,130);
x.showdetails();
cout << "\n Copy constructor output
student stu(x);
stu.showdetails();

}
```

* **Destructor** :- It is used to destroy the objects that have been created by constructor. A destructor destroy value of the object being destroyed. Destructor is called automatically by compiler. Destructor clean up storage that is no longer accessible.

Characteristics of destructor –

1- We can have only one constructor for a class. Destructor cannot be overloaded.
2- No argument can be provided to a destructor, neither does it return any value.
3- Destructor cannot be inherit.
4- A destructor may not be static.

Limitations of destructor –

1- These are case sensitive.
2- They are not good for programs having thousands lines of code.

Example :-

(14)

```
#include <iostream.h>
#include <conio.h>
class student
{
private:
```

```
int sid;
float marks;
public:
student (int i, float j)
{
sid = i;
marks = j;
 }
void showdetails()
{
cout<<"Student ID:"<<sid<<"Marks:"<<marks;
}
~student()
{
cout<<"Destructor is invoked";
}};
int main()
{
student s1(5,50.5);
s1.showdetails();
getch();
return 0;
 }
```

Output:-

Student ID: 5
Marks: 50.5
Destructor is invoked

* **Garbage collection** :- In C++ programming, garbage collection is a form of automatic memory management. The garbage collector or junk collector attempt to reclaim garbage or memory occupied by objects that are no longer in use by the program.

Garbage collection is essentially the opposite of manually memory management which requires the programmer to specify which object to deallocate and return to the memory system.

Ex: Destructor

* What is memory leak?

In C++, memory allocation that takes place at run time must be managed by the executing program itself. Typically run time memory is allocated on stack section for function and on heap section. There is no need to manage stack section, memory is allocated for the function call and is deallocated when the function returns.

Secondly the heap section memory is allocated using new operator and is release using delete operator.

Memory leak is a situation when executing progress is allocated memory from heap section but not release in stack.

Example of garbage collection -

```
#include <iostream.h>
#include <conio.h>
class mysample
{
public:
mysample()
{
cout<<" In constructor---"<<endl;
}
~mysample()
{
cout<<" In destructor---"<<endl;
}
};
void main()
{
mysample obj;
mysample *p = new mysample();
getch();
}
```

Output :- In constructor---
In destructor---

* Dynamic memory allocation :- Dynamic memory allocation allow to set array size dynamically during run time rather than at compile time.

This helps when the program does now know in advance about the number of items (variables - value to be stored). Dynamic memory allocation in C / C++ refer to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on heap and non-static and local variables get memory allocated on stack details.

How it is different from memory allocated to normal variables ?
For normal variable like int a, char, string, str[10] etc, memory is automatically and deallocated. For dynamically allocated memory like -

```
int *p = new int();
```

it is programmer's responsibility to deallocate memory, when no longer needed. It causes memory leak (memory is not deallocated until program terminate).

(6) Example of dynamic memory allocation -

```
#include <iostream.h>
#include <conio.h>
class cube
{
public:
cube()
```

```
{
cout<<"Constructor is called!"<<endl;
}
~cube()
{
cout<<"Destructor is called!"<<endl;
}
};
int main()
{
cbuser();
cube * mycubearray = new cube[3];
delete[] mycubearray;
getch();
return 0;
}
```

Output :- Constructor is called!
Destructor is called!

3

Note :- The new operator return a pointer to the object is allocated, the program must define a pointer with suitable scope to access those objects.

* **Abstract class :-** An abstract class is a class that is design to be specifically use as a base class. An abstract class contains at least one pure virtual function. You can declare a pure virtual function by using a pure specifier in the declaration of a virtual member function in the class declaration. Sometime implementation of all functions cannot be provided in a base class because we do not know the implementation such as a class is called abstract class.
An abstract class can have constructor. If we do not override the pure virtual function in derived class then derived class also become abstract class.
We can have pointers and refrence of abstract class type. A class is abstract if it has at least one pure virtual function.

(1.)

```
#include <iostream.h>
#include <conio.h>
class base
{
public:
virtual void disp() = 0;
};
class d : public base
```

```
{
public:
void disp()
{
cout <<"Derived class";
}
};
int main()
{
d obj;
obj.disp()
getch();
return 0;
}
```

Output :- Derived class

UNIT-3

* Access specifier :- There are three access specifier - public, private and protected. These access specifier define how the member of the class can be access ofcource any member of a class is accessible within that class inside any member function of that same class.

There are 3 types of access specifier -

1- Public - The member declare as a public are accessible from outside the class through an object of the class.

2- Protected - The members declared as a protected are accessible from outside the class but only if a class derived from it.

3- Private - These members are only accessible from within the class, no outside access is allowed.

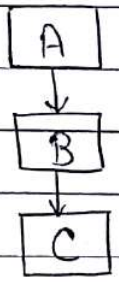
(127) Example of protected access specifier -

```
#include <iostream.h>
#include <conio.h>
class Shape
{
public:
void setwidth(int w)
```

```
{
width = w;
}
void setheight(int h)
{
height = h;
}
protected:
int width;
int height;
};
class Rectangle : public Shape
{
public:
int getarea()
{
return (width * height);
}
};
int main()
{
Rectangle R;
R.setwidth(6);
R.setheight(8);
cout<<" Area of rectangle :"<<R.getarea()<<endl;
getch();
return 0;
}
```

Output :- Area of rectangle : 48

* Multilevel Inheritance :- In multilevel inheritance, the class inherit the feature of another derived class.
In below diagram, class B and class A are as a parent classes depending on the relation. The level of inheritance can be extended any level.



(19) Example of multilevel inheritance -

```
class A
{
public:
int x;
void getdata()
{
cout << "Enter value of x:";
cin >> x;
}
};
class B : public A
{
public:
int y;
void readdata()
```

```
{
cout<<"Enter value of y:";
cin>>y;
}
};
class C : public B
{
private:
int z;
public:
void indata()
{
cout<<"\n Enter value of z:";
cin>>z;
}
void product()
{
cout<<"\n Product="<<x*y*z;
}
};
int main()
{
C a; // object of derived class
a.getdata;
a.readdata;
a.indata;
a.product;
getch();
return 0;
}
```

Output :- Enter value of x : 5
Enter value of y : 5
Enter value of z : 5
Product = 125.

* Hierarchical Inheritance :- In this type of inheritance more than one sub-class is inherited from a single base class. More than one derived class is created from a single base class.

Example :

```
# include < iostream.h >
# include < conio.h >
class A
{
public :
int x, y ;
void getdata ()
{
cout << "\n Enter value of x and y" ;
cin >> x >> y ;
}
} ;
class B : public A
{
public :
void product()
{
cout << "\n Product = " << x * y ;
```

```
}
};
class C : public A
{
public :
void sum()
{
cout << "\n sum = " << x+y;
}};
int main()
{
B obj1;
C obj2;
obj1.getdata();
obj1.product();
obj2.getdata();
obj2.sum();
getch();
return 0;
}
```

Output :-

```
Enter value of x and y:
6
5
Product = 30
Enter value of x and y:
6
6
sum = 12
```

* Polymorphism :- Polymorphism is the ability to use an operator or method in different ways. Polymorphism gives different meaning or function to the operator or methods.

Types of polymorphism – There are two types of polymorphism –

1- Compile time polymorphism
2- Run time polymorphism

1- Compile time polymorphism –
(i) Function overloading
(ii) Operator overloading

2- Run time polymorphism –
Function overriding / Virtual function

* Example of function overloading :-

```
class printdata
{
public :
void print (int i)
{
cout << "Print int:" << i << endl;
}
void print (double f)
{
cout << "Print float:" << f << endl;
```

```
}
void print (char *c)
{
cout <<" Print character :"<<c<< endl;
}};
int main()
{
printdata pd;
pd. print(5);
pd. print (500.263);
pd. print ("Hello C++");
getch ();
return 0;
  }
```

Output :-

Print int : 5
Print float : 500.263
Print character : Hello C++

* Function Overloading :- Function overloading refer to using the same thing for different purpose. This process of using two or more function with the same name but different in the signature, is called function overloading.

In term of type of argument.
In term of number of argument.

* Example of function overloading in term of number of argument -

```
class printdata
{
public:
void print (int i)
{
cout <<" Print int :" << i << endl;
}
void print (int b, int c)
{
cout <<" Print int:" <<b<<c << endl;
}
void print (int a, int b, int c)
{
cout <<" Print int:" <<a<<b<<c << endl;
}};
void main ()
{
printdata pd;
pd. print (5);
pd. print (6,6);
pd. print (7,8,9);
getch ();
}
```

Output :-

```
Print int: 5
Print int: 6 6
Print int: 7 8 9
```

* Function overriding — If derived class function name is same as that of base class member function with same argument then it is called as a function overriding.

```
class A
{
public:
void m1()
{
cout << "M1 of base";
};
class B: public A
{
void m1()
{
cout << "M1 of derived";
}};
int main()
{
 B b1;
 b1.m1;
 getch();
 return 0;
 }
```

Output :- M1 of derived

* Operator overloading :- C++ also provide option to overload operators.
For ex - we can make the operator '+' string class to concate two strings.
We know that it is the addition operator whose task is to add operands. So a single operator when place between integer operand add them and when place between string operand concate them.

// Using operator with built-in data type-

```
main()
{
int n1 = 5                    Output - 15
int n2 = 10
cout << " n1+n2;
}
```

Using operator with user define data type-

```
class test
{
int num;
public:
test()
{
num = 1;
};
```

```
main()
{
test t1,t2;
cout<< t1+t2;
  }
```

Ouput - Error

- ```
  class test
  {
  int num;
  public:
   test()
   {
  num=0;
   }
  void operator++()
   {
  ++num;
   }};
  main()
   {
  test t1 {
  ++t1;
   }}
  ```
  ```

* Member functions :- A member function perform an operation required by the class. It may be use to read, manipulate or display the data member. Member functions of a class can be define in

two place -
inside class definition, outside class definition.

* A program to understand concept of class & object-

```
# include < iostream.h >
# include < conio.h >
class employee
{
public :
int eid;
char name;
};
int main()
{
clrscr();
employee e1, e2;
e1. eid = 1;
e1. name = 'A';
e2. eid = 2;
e2. name = 'B';
cout << e1. eid << "---" << e1. name << "\n";
cout << e2. eid << "---" << e2. name << "\n";
getch();
return 0;
}
```

Output - 1---A
2---B

Program 7:- A program to access the values of private data members.

```
#include<iostream.h>
#include<conio.h>
class sum
{
private: int a,b,c;
public:
void add()
{
clrscr();
cout<<"Enter any two numbers:";
cin>>a>>b;
c=a+b;
cout<<"sum:"<<c;
}
};
void main()
{
sum s;
s.add();
getch();
}
```

Output - Enter any two numbers: 20
30
sum: 50

(A simple program based on encapsulation concept)

* (b) A program of classes and objects -

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class employee
{
public:
int eid;
char ename[3];
void enterdetails(int id, char name[])
{
eid = id;
strcpy(ename, name);
}
void showdetails()
{
cout<<"Employee ID"<<eid<<"Employee name"
                                <<ename<<endl;
}};
void main()
{
employee e1,e2;
e1.enterdetails(1,"ABC");
e2.enterdetails(2,"XYZ");
e1.showdetails();
e2.showdetails();
cout<<"Employee ID"<<e1.eid<<"---"<<"Employee name"
                                <<e1.ename<<"\n";
cout<<"Employee ID"<<e2.eid<<"---"<<"Employee name"
                                <<e2.ename<<"\n";
getch();
}
```

* Function :- A function is a group of statement that together perform a specific task. Every C++ program has at least one function which is main.

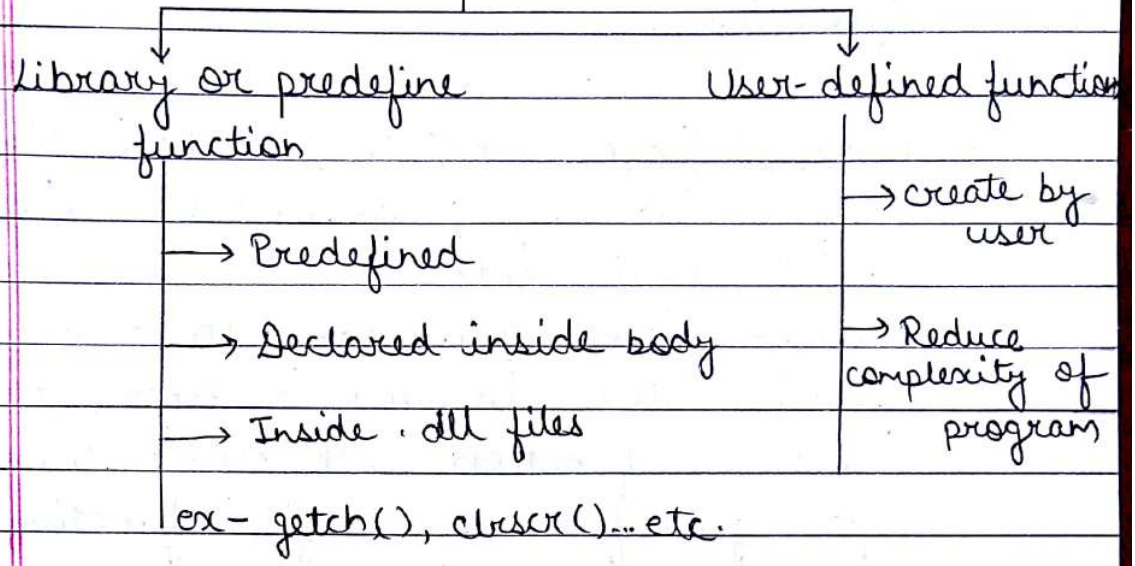
Why use function - Function are used for divide a large code into modules. Due to these we can easily debug and maintain the code.
For ex- If we write a calculator program At the time we can write every logic in a seperate function for addition (sum), subtraction. Any function can be call many time.

* Advantage of function -
1- Code reuse ability.
2- Develop an application in module format.
3- Easy to debug the program.
4- Code optimization, no need to write lot of code.

* Types of function :- There are two types of functions-

1) Library function or pre-define function.
2) User-defined function.

Function



* **Library function** :- Library functions are those, which are predefine in C++ compiler. The implementation part of predef-ine function is available in library files that are .lib /. obj files. .lib or .obj files contains predefine code.

Limitations of library functions –

1- All predefine functions are contained limited task only, that is for what porpose function is design for, same purpose it should be use.

2- As a programmer, we do not have any control on predefine functions. Implementa-tion part is there machine readable format.

3- In implementation whenever a predefine function is not supporting the user

requirement then go for user define functions.

* User-defined functions – These functions are created by programmers according to their requirements.
For ex- Suppose you want to create a function for add two numbers then you create a function with name sum. This type go for user define functions.

* Defining a function – Defining of function is nothing but give body to functions that means write logic inside function body.

1- Return type – A function may return a value. The return type is the value the function returns. Return type parameters and return statement are optional.

2- Function name – A function declaration is the process of tell the compiler about a function return.

3- Parameters – A parameter is like a place-holder. When a function is invoke you pass a value to the

parameters. This value is refer to the type ordered and number of the parameter of a function. Parameters are optional that is a function may contain number of parameters.

4- Function body – The function body contains a collection of statements that define what the function does.

5- Function Declaration – Function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be define seperately.
Syntax:

```
return_type functionname (Parameters);
```

6- Calling a function – When we call any function control goes to function body and execute entire code. For call any function just write name of function and if any parameter is required when pass parameter.

(8) For example:

```
#include <iostream.h>
#include <conio.h>
void sum();
```

```
int a = 10, b = 20, c;
void sum()
{
c = a + b;
cout << " sum:" << c;
}
void main()
{
clrscr();
sum();
getch();
}
```

Output :- sum: 30

→ calling function

* Namespace :- In C++, namespace are used to organise too many classes so that it can be easy to handle the applicatio for accessing the class of a namespace, we need to use namespace name, class name.

We can use it using keywords so that we don't have to use complete name all the time.

Example of namespace –

```
namespace first()
{
void sayhello()
{
cout<<"Hello first namespace"<<endl;
}}
namespace second()
{
void sayhello()
{
cout<<"Hello second namespace"<<endl;
}}
void main()
{
first :: sayhello();
second :: sayhello();
getch();
}
```

Output – Hello first namespace
Hello second namespace

* Exception handling — Exception handling in C++ is a process to handle run time errors. We perform exception handling so the normal flow of the application can be maintained even after errors.

All the exception classes in C++ are derived from std:: exception class. Let's see the list of C++ common exception classes.

1. Logic failure — It is an exception that can be detected by reading a code.

2. Runtime error — It is an exception that cannot be detected by reading a code.

3. Bed exception — It is used to hadle the unexpected exception in C++ program.

• C++ exception handling keywords — In C++ we use 3 keywords to perform exception handling.

1- try
2- catch
3- throw

C++ try/catch — Exception handling is performed using try/catch statement. The C++ try block is used to place the code

that may occur exception. The catch block is used to handle the exceptions
Ex- try / catch / block

try [] block – This block captures series of errors in any program at runtime and throw it to the catch block where user can customize the error message.

```
try
{

}
```

Catch [] block– This block catches the errors thrown by try block. This block contains method to customize error.
Syntax–

```
catch
{

}
```

throw function – This function is used to transfer the error from try block to catch block. This function plays major role to save program from crushing.

Ques1. Define primary and derived data types with example.

Ans. • Primary data type (built-in) –

1. Integer
2. Float
3. Char
4. Double

• Derived data types

1. Arrays
2. Functions
3. Pointers

Ques(2) Write a C++ program to print the first and last digit of a given number using do while loop.

Ans.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int lastdigit, firstdigit, no, copyno;
cout<<"Enter any num:";
cin>>no;
copyno=no;  // Gets last digit
lastdigit = no%10;
while(copyno>=10)
{
 copyno = copyno/10;
}
firstdigit = copyno;
```

```
cout<<"\n lastdigit"<< lastdigit;
cout<<"\n firstdigit"<< firstdigit;
getch();
}
```

Q2. Explain operator overloading mechanism to find the sum of 3 numbers.

2016

Q. Explain the terms: Class, exception handling, Call by value.

Ans. Call by value :- This approach is also popularly known as passing arguments by value. In this, approach the values of the actual arguments are passed to the function during function call. When control is transfer-red to the called function, the values of actual arguments are copied to the corresponding formal arguments and then, the body of the function is executed. It is the called function supposed to return a value, it is returned with the help of return statement.

Que Name any three library functions and any three preprocessor directives in C++.

Ans - Preprocessor directives :-

(a) Inclusion directives - This category has only one directive which is called # include. This inclusion directive is used to include files into the current file.

(b) Macro definition directives - These are used to define macros which are one or more program statements like functions and they are expended in ~~time~~ line. They

include #define & #undef.

(c) Conditional compilation directives — These are used to execute statements conditionally for debugging purposes, executing the code on different machine architectures etc. These this include if, #elif, #ifdef and #ifndef.

Ques Why do we need different access specifier in class?

Ans. Access specifiers in C++ class defines the access control rules. They are used to set boundaries for availaibility of members of class be if data members of member functions. They set the accessibility of classes, methods and other members. They are a specific part of programming lang.

Ques- W.A.P. in C++ to calculate and display area A and parameter P of a rectangle R using classes. Given that rectangle R of length l and breadth b, area = l*b and parameter P = a+(l+b).

Ans-
```
#include <iostream.h>
#include <math.h>
class rectangle
{
int length;
int breadth;
```

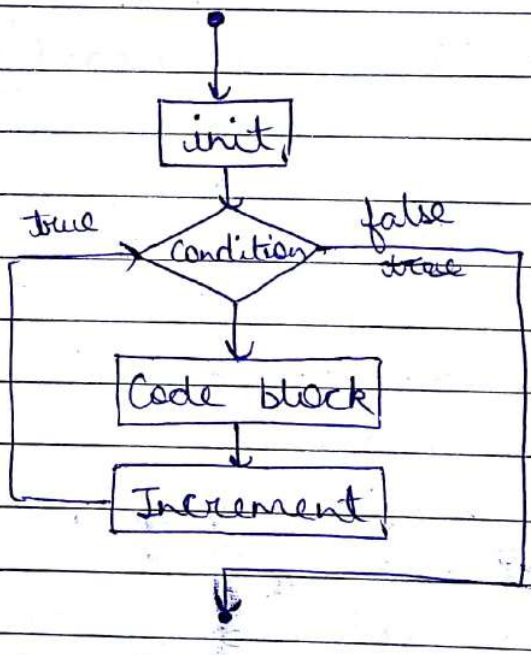
```
public :
  int getarea (int l, int b)
  {
  int area ;
  area = l * b ;
  cout<< " Area :" << area. <<"\n" ;
  }
int getperameter (int i ; int b)
  {
  int pert;
  pert =  +(l+b);
  cout<<" Parameter" <<part <<"\n";
  }
  },
  int main()
  {
  int l, b;
  rectangle R ;
  cout<<" Enter l of rect";
  cin >> l ;
  cout <<" Enter b of rect";
  cin >> b ;
  R. getarea ();
  R. getparameter ();
  return 0;
  }
```

Ques- Explain for loop in C++.

Ans- A for loop is a repetition control structure that allows us to efficiently write a loop that needs to execute a specific number of times.

Syntax:-

```
for (init. ; condition ; increment)
{
statement( );
}
```



Ex-

```
# include <iostream.h>
int main()
{
for( int a=10; a<20; a=a+1 )
{
cout << " Value of a :" << a << endl;
}
return 0;
}
```

Output – Value of 'a' : 10
" " " 11
" " 12
" " 19

Que Two single-D arrays A and B contain the elements as follows–
A[9] = 2,4,8,32,16,60,70,89,98
B[6] = 3,7,9,30,35,24
W.A.P that merges A and B and gives a third array C as follows: C[15] = 2,3,4,7,8–98

Ans.
```
# include <iostream.h>
# include <conio.h>
void main()
{
   int A[9], B[6], size1, size2, size, i, j, k, c[15];
   cout << "Enter array 1 size:";
   cin >> size1;
   cout << "Enter array 1 elements:";
   for (i=0; i<size1; i++)
   {
      cin >> A[i];
   }
   cout << "Enter array 2 size:";
   cin >> size2;
   cout << "Enter array 2 elements:";
   for (i=0; i<size2; i++)
   {
      cin >> B[i];
```

```
for(i=0; i<size1; i++)
 {
 c[i] = A[i];
 }
size = size1 + size2;
for(i=0; k=size1; k<size<
 {
  c[k]<B[i];
 }
cout<<"Now the new array after merging:"
for(i=0; i<size; i++)
 {
 cout<<c[i]<<" ";
 }
getch();
```

Ques - Discuss formatted and unformatted I/O opⁿs in stream classes.

Ans. C++ provides both the formatted and unformatted IO functions. In formatted or high-level IO, bytes are grouped and converted to types such as int, double, string or user-defined types. In unformatted or low-level IO, bytes are treated as raw bytes and unconverted. Formatted IO operations are supported via overloading the stream insertion (<<) and stream extraction (>>) operators, which presents a consistent public IO interfac

To perform input and output, a C++ program:

1- Constructs a stream object.
2- Connects the stream object to an actual IO device (e.g. keyboard, console, file network, another program).
3- Performs input/output operations on the stream, via the functions defined in the stream's public interface in a device independent manner.
4- Disconnects the stream to the actual IO device (ex- close the file).
5- Frees the stream object.

C++ IO is provided in headers <iostream>
The <iostream> header declares these standard stream objects.

(i) cin (of istream class), wcin (of wistream class)
(ii) cout (——————), wout (" " ")
(iii) cerr (——————), werr (——) standard error stream
(iv) clog (——), wclog (——), corresponding to the standard log stream, defaulted to display console.

Q Manipulators :-
Manipulators are helper functions that make it possible to control input/output stream using operator << or

operator >>. The manipulators that are invoked without arguments (ex. std::cout<< std::.boolalpha:; or std::cin >> std::hex;) are implemented as functions that takes a reference to a streams as their only argument.

Manipulators are stream functions available in "iomanip.h" headerfile and are used to change the default formats of input and output. These are used with stream insertion and extraction operators. They are are used for defining a specified format of input and output. They provide features similar to that of IOS member functions.

Some of the standard manipulators available in iomanip.h headerfile are endl, setw, setfill, hex, oct, dec, setprecision, flush, setioflags etc.

Manipulator are different from ios member functions as manipulator does not return the previous format state as is the case with ios member functions.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i,j,m,n,a[8][8], sum=0;
printf("Enter the order of the matrix:");
scanf("%d%d",&m,&n);
if(m==n)
{
printf("Enter elements in the matrix:%d",m*n);
for(i=0; i<m; i++)
{
for(j=0;j<n;j++)
}
printf("The enter matrix is:\n",m,n);
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%4d", a[i][j]);
if(i==j)
sum = sum + a[i][j];
}
printf("\n");
}
printf("Sum of diagonal elements of matrix is:%d\n", sum);
}
else
printf("Matrix should be a square matrix");
getch();
}
```

Ques. How delete[] is different from delete?

Ans. 1- delete is an operator whereas delete() is a library function.

2- delete free the allocated memory and calls destructor. But delete() de-allocate memory but does not call destructor.

3- delete is faster than delete() because an operator is always faster than a function.