

Lecture Notes
on
Object Oriented Programming
3CS4-06



Unit I

Department of Computer Science & Engineering
Jaipur Engineering College & Research Centre, Jaipur

Vision of the Institute

To become a renowned centre of outcome based learning and work toward academic, professional, cultural and social enrichment of the lives of individuals and communities.

Mission of the Institute

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.

M2: Identify, based on informed perception of Indian, regional and global needs, the areas of focus and provide platform to gain knowledge and solutions.

M3: Offer opportunities for interaction between academia and industry.

M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

Vision of the Department

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

Mission of the Department

M1: To impart outcome based education for emerging technologies in the field of computer science and engineering.

M2: To provide opportunities for interaction between academia and industry.

M3: To provide platform for lifelong learning by accepting the change in technologies.

M4: To develop aptitude of fulfilling social responsibilities.

Program Outcomes (PO)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and Computer Science & Engineering specialization to the solution of complex Computer Science & Engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex Computer Science and Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex Computer Science and Engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of Computer Science and Engineering experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex Computer Science Engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Computer Science and Engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional Computer Science and Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the Computer Science and Engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in Computer Science and Engineering.
10. **Communication:** Communicate effectively on complex Computer Science and Engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the Computer Science and Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change in Computer Science and Engineering.

Program Educational Objectives (PEO)

PEO1: To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science & Engineering by way of analyzing and exploiting engineering challenges.

PEO2: To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems in Computer Science and Engineering

PEO3: To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues for Computer Science & Engineering.

PEO4: To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career in Computer Science & Engineering.

PEO5: To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge in Computer Science & Engineering.

Course Outcomes

1. Understand the paradigms of object oriented programming in comparison of procedural oriented programming.
2. Apply the class structure as fundamental, building block for computational programming.
3. Apply the major object-oriented concepts to implement object oriented programs in C++.
4. Implement the concept of abstraction inheritance, polymorphism, dynamic binding and generic structure in building reusable code.

Mapping of Course Outcomes with Program Outcomes

H=3, M=2, L=1

Semester	Subject	Code	L / T / P	CO	P	P	P	P	P	P	P	P	P	P	P	P
					O	O	O	O	O	O	O	O	O	O	O	O
					1	2	3	4	5	6	7	8	9	10	11	12
III	Object Oriented Programming	3CS4-06	L	Understand the paradigms of object oriented programming in comparison of procedural oriented programming.	3	3	3	2	2	2	1	1	0	1	1	3
			L	Apply the class structure as fundamental, building block for computational programming.	3	3	3	3	2	2	1	1	1	2	1	3
			L	Apply the major object-oriented concepts to implement object oriented programs in C++.	3	3	3	3	2	1	2	2	1	2	1	3
			L	Implement the concept of abstraction inheritance, polymorphism, dynamic binding and generic structure in building reusable code.	3	3	3	2	1	1	1	1	1	1	1	3

Syllabus



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

II Year-III Semester: B.Tech. Computer Science and Engineering

3CS4-06: Object Oriented Programming

**Credit-3
3L+0T+0P**

**Max. Marks : 150 (IA:30,ETE:120)
End Term Exam: 3 Hours**

SN	CONTENTS	Hours
1	Introduction to different programming paradigm, characteristics of OOP, Class, Object, data member, member function, structures in C++, different access specifiers, defining member function inside and outside class, array of objects.	8
2	Concept of reference, dynamic memory allocation using new and delete operators, inline functions, function overloading, function with default arguments, constructors and destructors, friend function and classes, using this pointer.	8
3	Inheritance, types of inheritance, multiple inheritance, virtual base class, function overriding, abstract class and pure virtual function	9
4	Constant data member and member function, static data member and member function, polymorphism, operator overloading, dynamic binding and virtual function	9
5	Exception handling, Template, Stream class, File handling.	6
	TOTAL	40

UNIT I

Introduction:

Programmers write instructions in various programming languages to perform their computation tasks such as:

- (i) Machine level Language
- (ii) Assembly level Language
- (iii) High level Language

Machine level Language :

Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions.

Assembly level Language :

An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

High level Language :

High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture. High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada , Algol, BASIC, COBOL, C, C++, JAVA, FORTRAN, LISP, Pascal, and Prolog. Such languages are considered high-level because they are closer to human languages and farther from machine languages. In contrast, assembly languages are considered low-level because they are very close to machine languages.

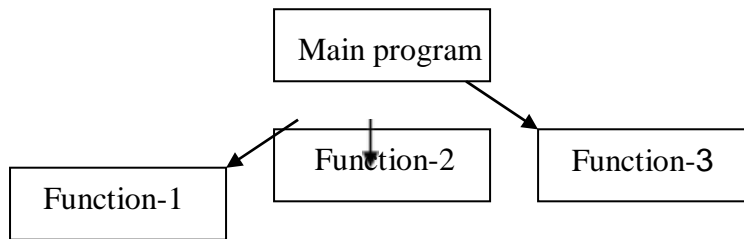
The high-level programming languages are broadly categorized in to two **programming paradigms:**

- (iv) Procedure oriented programming (POP) language.
- (v) Object oriented programming (OOP) language.

Procedure Oriented Programming Language

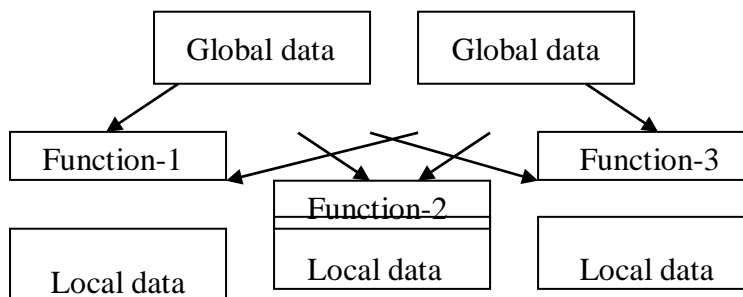
In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading , calculation and printing.

Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions.



The disadvantage of the procedure oriented programming languages is:

1. Global data access
2. It does not model real world problem very well
3. No data hiding

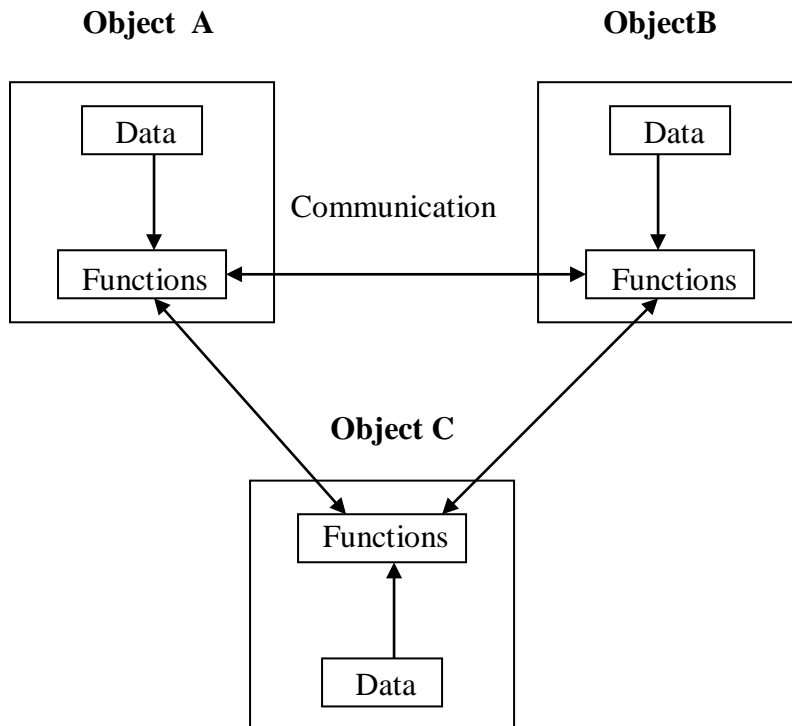


Characteristics of procedure oriented programming:

1. Emphasis is on doing things (algorithm)
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data
4. Data move openly around the system from function to function
5. Function transforms data from one form to another.
6. Employs top-down approach in program design

Object Oriented Programming

“Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand”.



Features of the Object Oriented programming

1. Emphasis is on doing rather than procedure.
2. programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure.
5. Data is hidden and can't be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added.
8. Follows bottom-up approach in program design.

BASIC CONCEPTS OF OBJECTS ORIENTED PROGRAMMING

1. Objects
2. Classes
3. Data abstraction and encapsulation
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing

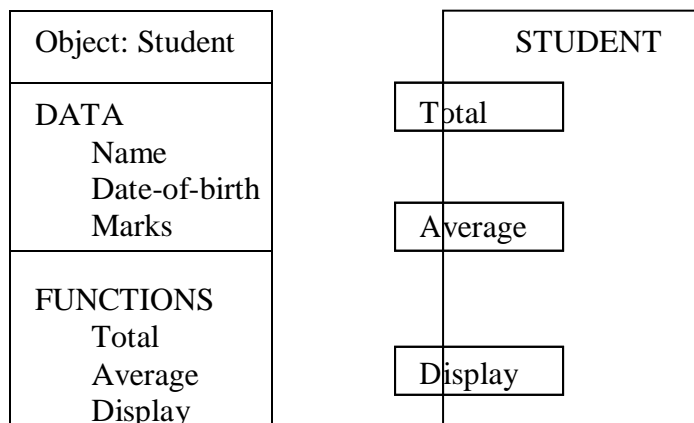
OBJECTS

Objects are the basic run-time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program must handle.

The fundamental idea behind object oriented approach is to combine both data and function into a single unit and these units are called objects.

The term objects means a combination of data and program that represent some real word entity. For example: consider an example named Amit; Amit is 25 years old and his salary is 2500. The Amit may be represented in a computer program as an object. The data part of the object would be (name: Amit, age: 25, salary:2500)

The program part of the object may be collection of programs (retrive of data, change age, change of salary). In general even any user –defined type-such as employee may be used. In the Amit object the name, age and salary are called attributes of theobject.



CLASS:

A group of objects that share common properties for data part and some program part are collectively called as class.

In C ++ a class is a new data type that contains member variables and member functions that operate on the variables.

DATA ABSTRACTION :

Abstraction refers to the act of representing essential features without including the back ground details or explanations. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

DATA ENCAPSALATION :

The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program.

INHERITENCE :

Inheritance is the process by which objects of one class acquire the properties of another class. In the concept of inheritance provides the idea of reusability. This mean that we can add additional features to an existing class with out modifying it. This is possible by desining a new class will have the combined features of both theclasses.

POLYMORPHISM:

Polymorphism means the ability to take more than one form. An operation may exhibit different instance. The behaviour depends upon the type of data used in the operation.

A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called determines which definition will be used.

Overloading may be operator overloading or function overloading.

It is able to express the operation of addition by a single operater say '+'. When this is possible you use the expression $x + y$ to denote the sum of x and y , for many different types of x and y ; integers , float and complex no. You can even define the $+$ operation for two strings to mean the concatenation of the strings.

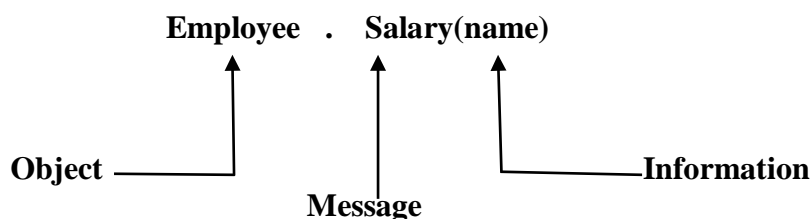
DYNAMIC BINDING :

Binding refers to the linking of a procedure call to the code to the executed in response to the call. Dynamic binding means the code associated with a given procedure call is not known untill the time of the call at run-time. It is associated with a polymorphic reference depends upon the dynamic type of that reference.

MESSAGE PASSING :

An object oriented program consists of a set of objects that communicate with each other.

A message for an object is a request for execution of a procedure and therefore will invoke a function (procedure) in the receiving object that generates the desired result. Message passing involves specifying the name of the object, the name of the function (message) and information to be sent.



BENEFITS OF OOP:

OOP offers several benefits to both the program designer and the user. Object-oriented contributes to the solution of many problems associated with the development and quality of software products. The principal advantages are:

1. Through inheritance we can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
3. This principle of data hiding helps the programmer to build secure programs that can't be invaded by code in other parts of the program.
4. It is possible to have multiple instances of an object to co-exist with out any interference.
5. It is easy to partition the work in a project based on objects.
6. Object-oriented systems can be easily upgraded from small to large systems.
7. Message passing techniques for communication between objects makes the interface description with external systems much simpler.
8. Software complexity can be easily managed.

APPLICATION OF OOP:

The most popular application of oops up to now, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using oop techniques.

Real business systems are often much more complex and contain many more objects with complicated attributes and methods. Oop is useful in this type of applications because it can simplify a complex problem. The promising areas for application of oop includes.

1. Real – Time systems.
2. Simulation and modeling
3. Object oriented databases.
4. Hypertext, hypermedia and expert text.
5. AI and expert systems.
6. Neural networks and parallel programming.
7. Decision support and office automation systems.
8. CIM / CAM / CAD system.

Basics of C++

C++ is an object oriented programming language, C++ was developed by Jarney Stroustrup at AT & T Bell lab, USA in early eighties. C++ was developed from c and simula 67 language. C++ was early called 'C with classes'.

C++ Comments:

C++ introduces a new comment symbol //(double slash). Comments start with a double slash symbol and terminate at the end of line. A comment may start any where in the line and what ever follows till the end of line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multi line comments can be written as follows:

```
// this is an example of  
// c++ program  
// thank you
```

The c comment symbols /**/ are still valid and more suitable for multi line comments.

```
/* this is an example of c++ program */
```

Output Operator:

The statement `cout <<"Hello, world"` displayed the string with in quotes on the screen. The identifier `cout` can be used to display individual characters, strings and even numbers. It is a predefined object that corresponds to the standard output stream. Stream just refers to a flow of data and the standard Output stream normally flows to the screen display. The `cout` object, whose properties are defined in `iostream.h` represents that stream. The insertion operator `<<` also called the 'put to' operator directs the information on its right to the object on its left.

Return Statement:

In C++ `main ()` returns an integer type value to the operating system. Therefore every `main ()` in C++ should end with a `return (0)` statement, otherwise a warning or an error might occur.

Input Operator:

```
The statement  
cin>> number 1;
```

is an input statement and causes. The program to wait for the user to type in a number. The number keyed in is placed in the variable `number1`. The identifier `cin` is a predefined object in C++ that corresponds to the standard input stream. Here this stream represents the key board.

The operator `>>` is known as get from operator. It extracts value from the keyboard and assigns it to the variable on its right.

Cascading Of I/O Operator:

```
cout<<"sum="<<sum<<"\n";
cout<<"sum="<<sum<<"\n"<<"average="<<average<<"\n";
cin>>number1>>number2;
```

Structure Of A Program :

Probably the best way to start learning a programming language is by writing a program. Therefore, here is our first program:

```
// my first program in C++
```

```
#include <iostream>
using namespace std;
```

```
int main ()
{
    cout << "Hello World!";
    return 0;
}
```

Output:-Hello World!

The first panel shows the source code for our first program. The second one shows the result of the program once compiled and executed. The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

The previous program is the typical program that programmer apprentices write for the first time, and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the code we have just written:

```
// my first program in C++
```

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

```
#include <iostream>
```

Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive `#include<iostream>` tells the preprocessor to include the `iostream` standard file. This specific file (`iostream`) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

```
using namespace std;
```

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name `std`. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

```
int main ()
```

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it – the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.

The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.

Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

cout << "Hello World!";

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen).

cout is declared in the iostream standard file within the std namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.

Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement).

return 0;

The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with directives for the compiler's preprocessor (those beginning by #). Then there were lines that began the declaration of a function (in this case, the main function) and, finally lines with statements (like the insertion into cout), which were all included within the block delimited by the braces ({}) of the main function.

The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

```
int main ()
{
    cout << " Hello World!";
    return 0;
}
```

We could have written:

```
int main ()
{
    cout << "Hello World!";
}
```

```
return 0;
}
```

All in just one line and this would have had exactly the same meaning as the previous code.

In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:

```
// my second program in C++
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World! ";
    cout << "I'm a C++ program";
    return 0;
}
```

Output:-Hello World! I'm a C++ program

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since main could have been perfectly valid defined this way:

```
int main ()
{
    cout << " Hello World! ";
    cout << " I'm a C++ program ";
    return 0;
}
```

We were also free to divide the code into more lines if we considered it more convenient:

```
int main ()
{
    cout <<"Hello World!"; cout
    << "I'm a C++program";
    return0;
}
```

And the result would again have been exactly the same as in the previous examples.

Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

STRUCTURE OF C++ PROGRAM

- Include files
- Class declaration
- Class functions, definition

- Main function program

Example :-

```
# include<iostream.h>

class person
{
char name[30];
int age;
public:
    void getdata(void);
    void display(void);
};

void person :: getdata ( void )
{
    cout<<"enter name";
    cin>>name;
    cout<<"enter age";
    cin>>age;
}

void display()
{
    cout<<"\n name:"<<name;
    cout<<"\n age:"<<age;
}

int main()
{

    person p;
    p.getdata();
    p.display();
    return(0);

}
```

TOKENS:

The smallest individual units in program are known as **tokens**. C++ has the following tokens.

- i. Keywords
- ii. Identifiers
- iii. Constants
- iv. Strings
- v. Operators

KEYWORDS:

The keywords implement specific C++ language feature. They are explicitly reserved identifiers and can't be used as names for the program variables or other user defined program elements. The keywords not found in ANSI C are shown in red letter.

C++ KEYWORDS:

Asm	double	new	switch
Auto	else	operator	template
Break	enum	private	this
Case	extern	protected	throw
Catch	float	public	try
Char	for	register	typedef
Class	friend	return	union
Const	goto	short	unsigned
Continue	if	signed	virtual
Default	inline	sizeof	void
Delete	long	struet	while

IDENTIFIERS:

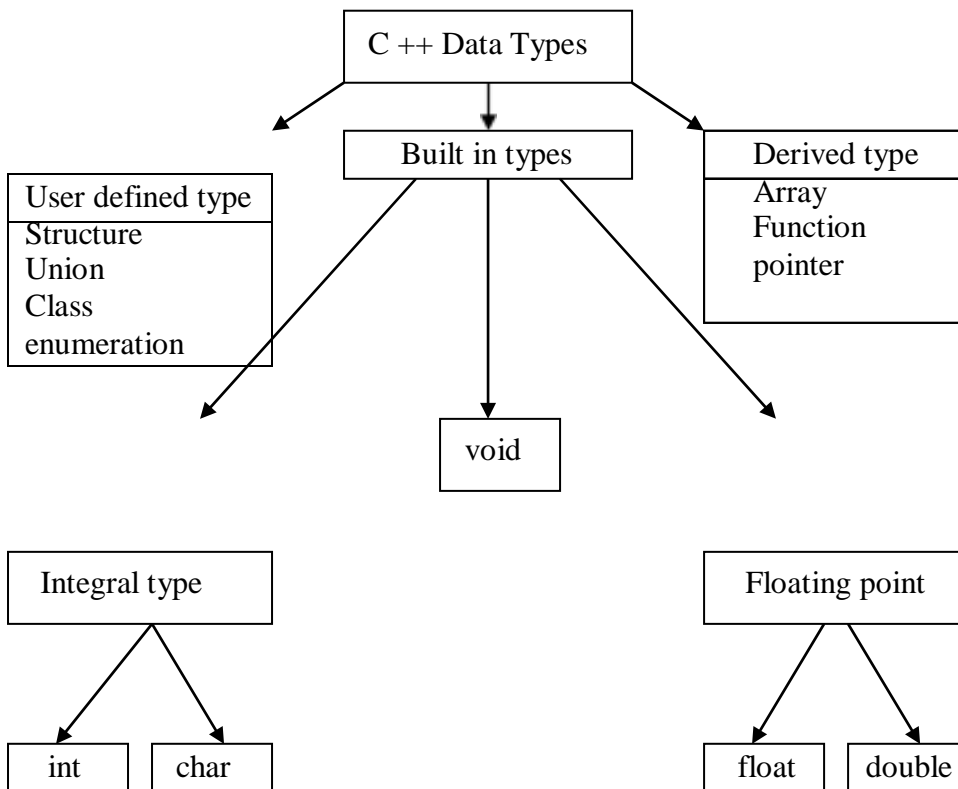
Identifiers refers to the name of variable , functions, array, class etc. created by programmer. Each language has its own rule for naming the identifiers.

The following rules are common for both C and C++.

1. Only alphabetic chars, digits and under score arepermitted.
2. The name can't start with adigit.
3. Upper case and lower case letters aredistinct.
4. A declared keyword can't be used as a variablename.

In ANSI C the maximum length of a variable is 32 chars but in c++ there is no bar.

BASIC DATA TYPES IN C++



Both C and C++ compilers support all the built in types. With the exception of void the basic datatypes may have several modifiers preceding them to serve the needs of various situations. The modifiers signed, unsigned, long and short may applied to character and integer basic data types. However the modifier long may also be applied to double.

Data types in C++ can be classified under various categories.

<u>TYPE</u>	<u>BYTES</u>	<u>RANGE</u>
char	1	-128 to – 127
unsigned	1	0 to 265
sgned char	1	-128 to 127
int	2	-32768 to 32768
unsigned int	2	0 to 65535
singed int	2	-32768 to 32768
short int	2	-32768 to 32768

long int	4	-2147483648 to 2147483648
signed long int	4	-2147483648 to 2147483648
unsigned long int	4	0 to 4294967295
float	4	3.4E-38 to 3.4E+38
double	8	1.7E -308 to 1.7E +308
long double	10	3.4E-4932 to 1.1E+ 4932

The type void normally used for:

- 1) To specify the return type of function when it is not returning anyvalue.
- 2) To indicate an empty argument list to afunction.

Example:

Void function(void);

Another interesting use of void is in the declaration of genetic pointer

Example:

Void *gp;

Assigning any pointer type to a void pointer without using a cast is allowed in both C and ANSI C. In ANSI C we can also assign a void pointer to a non-void pointer without using a cast to non void pointer type. This is not allowed in C++.

Example:

void *ptr1;

void *ptr2;

Are valid statement in ANSI C but not in C++. We need to use a cast operator.

ptr2=(char *) ptr1;

USER DEFINED DATA TYPES:

STRUCTERS AND CLASSES

We have used user defined data types such as struct,and union in C. While these more features have been added to make them suitable for object oriented programming. C++ also permits us to define

another user defined data type known as class which can be used just like any other basic data type to declare a variable. The class variables are known as objects, which are the central focus of oops.

ENUMERATED DATA TYPE:

An enumerated data type is another user defined type which provides a way for attaching names to number, these by increasing comprehensibility of the code. The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and soon. This facility provides an alternative means for creating symbolic.

Example:

```
enum shape {circle,square,triangle}
```

```
enum colour{red,blue,green,yellow}
```

```
enum position{off,on}
```

The enumerated data types differ slightly in C++ when compared with ANSI C. In C++, the tag names shape, colour, and position become new type names. That means we can declare new variables using the tag names.

Example:

```
Shape ellipse;//ellipse is of type shape
```

```
colour background ; // back ground is of type colour
```

ANSI C defines the types of enums to be ints. In C++,each enumerated data type retains its own separate type. This means that C++ does not allow an int value to be automatically converted to an enum.

Example:

```
colour background =blue; //vaid
```

```
colour background =7; //error in c++
```

```
colour background =(colour) 7;//ok
```

How ever an enumerated value can be used in place of an int value.

Example:

```
int c=red ;//valid, colour type promoted to int
```

By default, the enumerators are assigned integer values starting with 0 for the first enumerator, 1 for the second and so on. We can also write

```
enum color { red, blue=4,green=8};
```

```
enum color {red=5,blue,green};
```

C++ also permits the creation of anonymous enums (i.e, enums without tag names)

Example:

```
enum{off,on};
```

Here off is 0 and on is 1.these constants may be referenced in the same manner as regular constants.

Example:

```
int switch-1=off;
```

```
int switch-2=on;
```

ANSI C permits an enum defined with in a structure or a class, but the enum is globally visible. In C++ an enum defined with in a class is local to that class.

SYMBOLIC CONSTANT:

There are two ways of creating symbolic constants in c++.

1. using the qualifier const.
2. defining a set of integer constants using enum keywords.

In both C and C++, any value declared as const can't be modified by the program in any way. In C++, we can use const in a constant expression. Such as

```
const int size = 10 ;  
char name (size) ;
```

This would be illegal in C. const allows us to create typed constants instead of having to use #define to create constants that have no type information.

```
const size=10;
```

Means

```
const int size =10;
```

C++ requires a const to be initialized. ANSI C does not require an initializer, if none is given, it initializes the const to 0.

In C++const values are local and in ANSIC const values are global. However they can be made local made local by declaring them as static. In C++ if we want to make const value as global then declare as extern storage class.

Ex: external consttotal=100;
of naming integer constants is as follows:-

Another method

```
enum {x,y,z};
```

DECLARATION OF VARIABLES:

In ANSIC C all the variable which is to be used in programs must be declared at the beginning of the program. But in C++ we can declare the variables anywhere in the program where it requires. This makes the program much easier to write and reduces the errors that may be caused by having to scan back and forth. It also makes the program easier to understand because the variables are declared in the context of their use.

Example:

```
main()
{
    float x,average;
    float sum=0;
    for(int i=1;i<5;i++)
    {
        cin>>x;
        sum=sum+x
    }
    float average;
    average=sum/x;
    cout<<average;
}
```

REFERENCE VARIABLES:

C++ interfaces a new kind of variable known as the reference variable. A reference variable provides an alias (alternative name) for a previously defined variable. For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent the variable.

A reference variable is created as follows:

Syntax: Datatype & reference –name=variable name;

Example:

```
float total=1500;
float &sum=total;
```

Here sum is the alternative name for variable total, both the variables refer to the same data object in the memory.

A reference variable must be initialized at the time of declaration.

Note that C++ assigns additional meaning to the symbol & here & is not an address operator. The notation float & means reference to float.

Example:

```
int n[10];
int &x=n[10];
char &a='\n';
```

OPERATORS IN C++ :

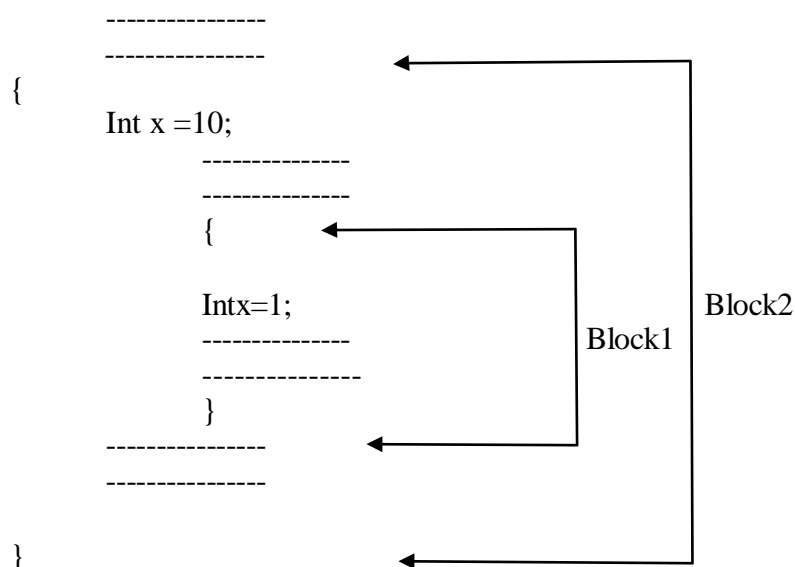
C++ has a rich set of operators. All C operators are valid in C++ also. In addition, C++ introduces some new operators.

<<	insertion operator
>>	extraction operator
::	scope resolution operator
::*	pointer to member declarator
*	pointer to member operator
.*	pointer to member operator
Delete	memory release operator
endl	line feed operator
New	memory allocation operator
setw	field width operator

SCOPE RESOLUTION OPERATOR:

Like C, C++ is also a block-structured language. Blocks and scopes can be used in constructing programs. We know some variables can be declared in different blocks because the variables declared in blocks are local to that function.

Blocks in C++ are often nested. Example:



Block2 contained in block 1 .Note that declaration in an inner block hides a declaration of the same variable in an outer block and therefore each declaration of x causes it to refer to a different data object . With in the inner block the variable x will refer to the data object declared there in.

In C, the global version of a variable can't be accessed from within the inner block. C++ resolves this problem by introducing a new operator :: called the scope resolution operator. This can be used to uncover a hidden variable.

Syntax: :: variable-name;

Example:

```
#include <iostream.h>
int m=10;
main()
{
int m=20;
{
int k=m;
int m=30;
cout<<"we are in inner block";
cout<<'k='<<k<<endl;
cout<<'m='<<m<<endl;
cout<<":: m="<<::m<<endl;
}
cout<<"\n we are in outer block\n";
cout<<'m='<<m<<endl;
cout<<":: m="<<::m<<endl;
}
```

Memory Management Operator

C uses malloc and calloc functions to allocate memory dynamically at run time. Similarly it uses the functions free() to free dynamically allocated memory. We use dynamic allocation techniques when it is not known in advance how much of memory space is needed.

C++ also supports those functions; it also defines two unary operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

The new operator can be used to create objects of any type.

Syntax:

pointer- variable =new datatype;

Example:

p=new int; q=new int;

Where p is a pointer of type int and q is a pointer of type float.

int *p=new int;

float *p=new float;

Subsequently, the statements

***p=25;**


```
*q=7.5;
```

Assign 25 to the newly created int object and 7.5 to the float object. We can also initialize the memory using the new operator.

Syntax:

```
int *p=new int(25);  
float *q =new float(7.5);
```

new can be used to create a memory space for any data type including user defined such as arrays, structures, and classes. The general form for a one-dimensional array is:

```
pointer-variable =new data types [size];
```

creates a memory space for an array of 10 integers.

If a data object is no longer needed, it is destroyed to release the memory space for reuse.

Syntax: delete pointer-variable;

Example:

```
delete p;  
delete q;
```

If we want to free a dynamically allocated array, we must use the following form of delete.

```
delete [size] pointer-variable;  
or
```

```
delete [ ] pointer variable;
```

MANIPULATORS:

Manipulators are operators that are used to format the data display. The most commonly used manipulators are endl and setw.

The endl manipulator, when used in an output statement, causes a line feed to be inserted (just like \n).

Example:

```
cout<<"m="<<m<<endl;  
cout<<"n="<<n<<endl;  
cout<<"p="<<p<<endl;
```

If we assume the values of the variables as 2597, 14 and 175 respectively

```
m=2597; n=14;
```

```
p=175
```

It was wanted to print all nos in right justified way using setw which specifies a common field width for all the nos.

```
Example: cout<<setw(5)<<sum<<endl;  
cout<<setw(10)<<"basic"<<setw(10)<<basic<<endl;  
cout<<setw(10)<<"allowance"<<setw(10)<<allowance<<endl;  
cout<<setw(10)<<"total="<<setw(10)<<total;
```

CONTROL STRUCTURES:

Like c, c++, supports all the basic control structures and implements them various control statements.

The if statement:

The if statement is implemented in two forms:

1. simple if statement
2. if... else statement

Simple if statement:

if (condition)

{

Action;

}

If.. else statement

If (condition)

Statement1

Else

Statement2

The switch statement

This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points;

Switch(expr)

{

case 1:

action1;

break;

case 2:

action2;

break;

..

..

default:

message

}

The while

statement:

While(condition)

{

Stements

}

The do-while statement:

Syn:

```
do  
  
{  
  
Statements  
  
} while(condition);
```

The for loop:

```
for(expression1;expression2;expression3)  
  
{  
  
Statements;  
  
Statements;  
  
}
```

FUNCTION IN C++ :

The main() Functon :

ANSI does not specify any return type for the main () function which is the starting point for the execution of a program . The definition of main() is :-

```
main()  
{  
//main program statements  
}
```

This is property valid because the main() in ANSIC does not return any value. In C++, the main() returns a value of type int to the operating system.The functions that have a return value should use the return statement for terminating. The main()function in C++ is therefore defined as follows.

```
int main( )  
{  
-----  
-----  
return(0)  
}
```

Since the return type of functions is int by default, the key word int in the main() header is optional.

INLINE FUNCTION:

To eliminate the cost of calls to small functions C++ proposes a new feature called inline function. An inline function is a function that is expanded inline when it is invoked. That is the compiler replaces the function call with the corresponding function code.

The inline functions are defined as follows:-

```
inline function-header
{
function body;
}
```

Example: inline double cube (double a)
{
return(a*a*a);
}

The above inline function can be invoked by statements like

```
c=cube(3.0);
d=cube(2.5+1.5);
```

remember that the inline keyword merely sends a request, not a command to the compiler. The compiler may ignore this request if the function definition is too long or too complicated and compile the function as a normal function.

CLASS:-

Class is a group of objects that share common properties and relationships. In C++, a class is a new data type that contains member variables and member functions that operates on the variables. A class is defined with the keyword class. It allows the data to be hidden, if necessary from external use. When we defining a class, we are creating a new abstract data type that can be treated like any other built in datatype.

Generally a class specification has two parts:-

- a) Class declaration
- b) Class function definition

the class declaration describes the type and scope of its members. The class function definition describes how the class functions are implemented.

Syntax:-

```
class class-name
{
private:
variable declarations;
function declaration ;
public:
variable declarations;
function declaration;
};
```

The members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also. The data hiding is the key feature of oops. The use of keywords private is optional by default, the members of a class are private. The variables declared inside the class are known as data members and the functions are known as members and the functions. Only the member functions can have access to the private data members and private functions. However, the public members can be accessed from the outside the class. The binding of data and functions together into a single class type variable is referred to as encapsulation.

Syntax:-

```
class item
{
    int member;
    float cost;
public:
    void getldata (int a ,float b);
    void putdata (void);
```

The class item contains two data members and two function members, the data members are private by default while both the functions are public by declaration. The function getldata() can be used to assign values to the member variables member and cost, and putdata() for displaying their values . These functions provide the only access to the data members from outside the class.

CREATING OBJECTS:

Once a class has been declared we can create variables of that type by using the classname.

Example:

```
item x;
```

creates a variables x of type item. In C++, the class variables are known as objects. Therefore x is called an object of type item.

item x, y ,z also possible.

```
class item
```

```
{
```

```
-----
```

```
-----
```

```
-----
```

```
}x ,y ,z;
```

would create the objects x ,y ,z of type item.

ACCESSING CLASS MEMBER:

The private data of a class can be accessed only through the member functions of that class. The main() cannot contains statements that the access number and cost directly.

Syntax:

```
object name.function-name(actualarguments);
```

Example:- x.getdata(100,75.5);

It assigns value 100 to number, and 75.5 to cost of the object x by implementing the getdata() function .

similarly the statement

```
x. putdata ( ); //would display the values of data members.
```

x. number = 100 is illegal .Although x is an object of the type item to which number belongs , the number can be accessed only through a member function and not by the objectdirectly.

Example:

```
class xyz
```

```
{
```

```
    Intx;
```

```
    Inty;
```

```
public:
```

```
    intz;
```

```
};
```

```
-----
```

```
-----
```

```
xyz p;
```

```
p.x=0;
```

```
p,z=10;
```

error . x isprivate

ok ,z ispublic

DEFINING MEMBER FUNCTION:

Member can be defined in two places

- Outside the class definition
- Inside the classfunction

OUTSIDE THE CLASS DEFINATION:

Member function that are declared inside a class have to be defined separately outside the class. Their definition are very much like the normal functions.

An important difference between a member function and a normal function is that a member function incorporates a membership. Identify label in the header. The 'label' tells the compiler which class the function belongs to.

Syntax:

```
return type class-name::function-name(argument declaration )
    {
        function-body
    }
```

The member ship label class-name :: tells the compiler that the function function - name belongs to the class class-name . That is the scope of the function is restricted to the class-name specified in the header line. The :: symbol is called scope resolution operator.

Example:

```
void item :: getdata (int a , float b )
    {
        number=a;
        cost=b;
    }
void item :: putdata ( void)
    {
        cout<<"number="<<number<<endl;
        cout<<"cost="<<cost<<endl;
    }
```

The member function have some special characteristics that are often used in the program development.

- Several different classes can use the same functionname. The "membership label" will resolve their scope, member functions can access the private data of the class . A non member function can't do so.
- A member function can call another member function directly, without using the dot operator.

INSIDE THE CLASS DEFINATION:

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

Example:

```
class item
{
    int number;
    float cost;
public:
    void getdata (int a ,float b);
    void putdata(void)
    {
        cout<<number<<endl; cout<<cost<<endl;
    }
};
```

A C++ PROGRAM WITH CLASS:

```
#include< iostream. h>
class item
{
    int number;
    float cost;
public:
    void getdata ( int a , float b);
    void putdala ( void)
    {
        cout<<"number:"<<number<<endl;
        cout<<"cost :"<<cost<<endl;
    }
};

void item :: getdata (int a , float b)
{
    number=a;
    cost=b;
}

main ( )
{
    item x;
    cout<<"\nobjectx"<<endl;
    x. getdata( 100,299.95);
    x .putdata();
    item y;
    cout<<"\n object y"<<endl;
    y. getdata(200,175.5);
    y. putdata();
}
```

Output: **object x**
 number100

cost=299.950012
object -4
cost=175.5

Q.

Write a simple program using class in C++ to input subject mark and printsit.

ans:

```
class marks
{
    private :
        int ml,m2;
    public:
        void getdata();
        void displaydata();
};
void marks: :getdata()
{
    cout<<"enter 1st subject mark:";
    cin>>ml;
    cout<<"enter 2nd subject mark:";
    cin>>m2;
}
void marks: :displaydata()
{
    cout<<"1st subject mark:"<<ml<<endl ;
    cout<<"2nd subject mark:"<<m2;
}
voidmain()
{
    clrscr();
    marks x;
    x.getdata();
    x.displaydata();
}
```

NESTING OF MEMBER FUNCTION:

A member function can be called by using its name inside another member function of the same class. This is known as nesting of member functions.

```
#include <iostream.h>
class set
{
    int m,n;
    public:
        void input(void);
        void display (void);
        void largest(void);
};
int set::largest (void)
{
    if(m>n)
        return m;
    else
        return n;
}
void set::input(void)
{
    cout<<"input values of m and n:";
    cin>>m>>n;
}
void set::display(void)
{
    cout<<"largestvalue="<<largest()<<"\n";
}
void main()
{
    set A;
    A.input( );
    A.display( );
}
```

output:

Input values of m and n:

3017

largest value= 30

Private member functions:

Although it is a normal practice to place all the data items in a private section and all the functions in public, some situations may require contain functions to be hidden from the outside calls. Tasks such as deleting an account in a customer file or providing increment to and employee are events of serious consequences and therefore the functions handling such tasks should have restricted access. We can place these functions in the privatesection.

A private member function can only be called by another function that is a member of its class. Even an object can not invoke a private function using the dot operator.

Class sample

```
{  
    intm;  
    void read (void);  
    void write (void);  
};
```

if si is an object of sample, then

```
s.read();
```

is illegal. How ever the function read() can be called by the function update () to update the value of m.

```
void sample :: update(void)  
    {  
        read( );  
    }
```

```

#include<iostream.h>

class part
{
private:
    intmodelnum,partnum;
    float cost;
public:
    void setpart ( int mn, int pn ,float c)
    {
        modelmim=mn;
        partnum=pn;
        cost=e;
    }
    void showpart ( )
    {
        Cout<<endl<<"model:"<<modelnum<<endl;
        Cout<<"num:"<< partnum <<endl
        Cout<<"cost:"<<"$"<<cost;
    }
};
void main()
{
    part p1,p2;
    p1.setpart(644,73,217.55);
    p2.setpart(567,89,789.55);
    p1.showpart();
    p1.showpart();
}

```

output:- model:644

num:73

cost: \$217550003

model: 567

num:89

cost: \$759.549988

```

#include<iostream.h>
Class distance
{
private:
    int feet;
    float inches;
public:
    void setdist ( int ft, float in)
        {
            feet=ft;
            inches=in;
        }
    void getdist()
    {
        cout<<"enter feet:";
        cin>>feet;
        cout<<"enter inches:";
        cin>>inches;
    }
    void showdist()
    {
        cout<< feet<<"' "inches<<endl;
    }
};
void main()
{
    distance d1,d2;
    d1.setdist(11,6.25);
    d2.getdata();
    cout<<endl<<"dist:"<<d1.showdist();
    cout<<"\n"<<"dist2:";
    d2.showdist();
}

```

output:- enter feet: 12
 enter inches:6.25
 dist 1:"11'-6.15"
 dist 2: 12'-6.25"

ARRAY WITH CLASSES:

```
#include<iostream.h>
#include<conio.h>
class employee
{
private:
    char name[20];
    int age,sal;
public:
    void getdata();
    void putdata();
};
void employee :: getdata ()
{
    cout<<"enter name :";
    cin>>name;
    cout<<"enter age :";
    cin>>age;
    cout<<"enter salary:";
    cin>>sal;
    return(0);
}
void employee :: putdata ()
{
    cout<<name <<endl;
    cout<<age<<endl;
    cout<<sal<<endl;
    return(0);
}
intmain()
{
```

```

employee emp[5]:
for( int i=0;i<5;i++)
{
emp[i].getdata();
}
cout<<endl;
for(i=0;i<5;i++)
{
emp[i].putdata();
}
getch();
return(0);
}

```

ARRAY OF OBJECTS:-

```

#include<iostream.h>
#include<conio.h>
class emp
{
private:
char name[20];
int age,sal;
public:
void getdata( );
void putdata( );
};

void emp :: getdata()
{
cout<<"enter empname": .
cin>>name;
cout<<"enter age:"<<endl;
cin>>age;
cout<<"enter salun :";

```



```

        cin>>sal;
    }
void emp :: putdata()
{
    cout<<"emp name:"<<name<<endl;
    cout<<"emp age:"<<age<<endl;
    cout<<"emp salary:"<<sal;
}

void main()
{
    emp   foreman[5];
    emp   engineer[5];
    for(int i=0;i<5;i++)
    {
        cout<<" for foreman:";
        foreman[i] . getdata();
    }
    cout<<endl;
    for(i=0;i<5;i++)
    {
        Foreman[i].putdata(); .
    }
    for(int i=0;i<5;i++)
    {
        cout<<" for engineer:";
        ingineer[i].getdata();
    }
    for(i=0;i<5;i++)
    {
        ingineer[i].putdata();
    }
    getch();
    return(0)
}

```