C++ PROGRAMMING

UNIT - 1

→ Programming Languages
→ Structured Prog. Paradigm
→ Object Oriented Prog. Paradigm
→ Structured Vs. Object Oriented Programming
→ Object Oriented Programming Concepts

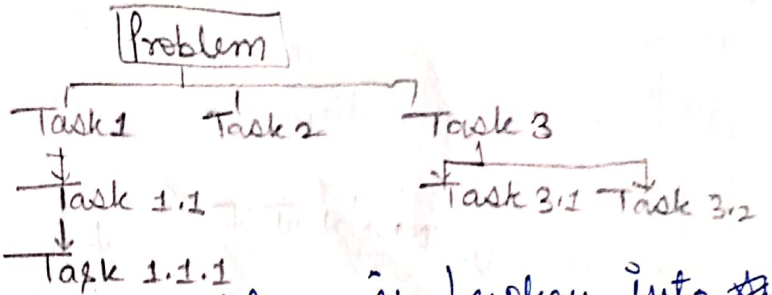
Structured Programming

In the process of coding, the lines of code keep multiplying, thus, size of the s/w increases. If one forgets how s/w & its underlying programs, files, procedures are constructed, it then becomes very difficult to share, debug and modify the program. The solution to this is Structured Programming.

→ It follows the Principle of divide and Conquer.
→ Use of GOTO stmts is completely forbidden in lang. supporting structured programming, brings clarity of code & improving efficiency.
→ Also helps programmer to reducing coding time & organise code properly.
→ Some of languages that follow this paradigm are C, C++, Java etc.
→ Uses 3 main approaches.
  (1) Top-down analysis
  (2) Modular Programming
  (3) Structured Coding

Top-down Analysis -

- Subdivision of Program
- Hierarchy of tasks



In Top-down analysis, the problem is broken into small pieces where each one has some significance. Each problem is individually solved & steps are clearly stated about how to solve the problem.

Modular Programming

→ While programming, the code is broken down into smaller group of instructions. These groups are known as modules, subprograms or subroutines.

→ Each module is developed independent of each other.

→ " " accomplishes one module function and contain all the source code.

```
main()  } 1 function
{
---
module 1
{
---     } 1 function
module 2
{
---     } 1 function
}
```

→ It leads to reusable code that can be used in other programs too.

→ Debugging & finding errors are easy.

Structured Coding

→ In reference with top-down analysis, the structured coding subdivides the modules into further smaller units of codes in order of their execution.

→ It uses control structure, w/c controls the flow of the program. Old Languages ↓ goto } unstructured code

goto can be replaced by if, if-else, else-if to write the code in structured manner.

→ It improves prob. solving process.
→ Better organisation of program.
→ Generalised programming methodology.
→ Clear description of data & structure.
} Advantages

Advantages of Structured Programming

(1) Complexity can be reduced using the concepts of divide and conquer.
(2) Logical structures ensures clear flow of control.
(3) Modules can be re-used many times, thus it saves time, reduces complexity, and increases reliability.
(4) Easier to update/fix the prog. by replacing individual modules rather than larger amounts of code.
(5) Ability to either eliminate or at least reduce the necessity of employing GOTO statement.

Disadvantages -

(1) Since GOTO statement is not used, the structure of the program needs to be planned meticulously. (पूरी बारीकी से)
(2) Lack of Encapsulation
(3) Lack of Information hiding.
(4) Change of even a single data structure in a program necessitates changes at many places throughout it.

OBJECT ORIENTED PROGRAMMING

Introduction - OOP is programming paradigm that represents concepts as objects that has data fields & associated procedures known as methods.

→ Everything in OOP is grouped as self sustainable "objects".

→ There are several languages we are following the OOPs concept -

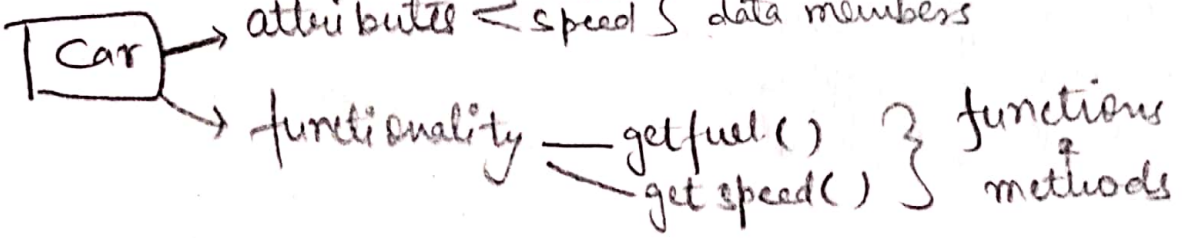
- Java
- Python
- Ruby
- C++
- Smalltalk
- Visual BASIC.NET
- Curl
- Delphi
- Effel

→ It trying to simulate real life things into objects our programming.

→ OOP was introduced to overcome flaws in the procedural approach to programing such as. Reusability and Maintainability.

Objects & Classes (collection of objects)

Object :- It is a piece of code w/c represent the real life entity.



Class : Class is used to represent a set of objects having same structure & behaviour.

```
class student
{
    R-NO ;
    Name ;
    marks ;
}
```

specifications of object

```
Student s ;
```

class ↑ object

Structured Programming Vs OOPS

Structured Programming	Object Oriented Programming
① It is intended w/c gives importance on logical structure or process and also emphasis on the data w/c is required for the process.	It emphasis on data.
② It is a Top-down approach.	It is a Bottom-up approach.
③ Programs are divided into small units called functions.	Programs are divided into small run time entities called objects.
④ Structure programming can solve simple programs. It cannot handle complex programs	OOP are designed to solve any level of complexity.
⑤ As no data hiding. Feature in S.P. is less secure.	Data hiding feature in OOP makes programming more secure.
⑥ S.P. is a subset of procedural prog-lang. & is also known as Modular Programming.	OOP supports paradigms encapsulation, data hiding, data binding, inheritance, abstraction, encapsulation.
⑦ Structured Prog. facilitates no reusability & also has func^n. dependency to large extent.	OOP facilitates reusability & also has less function dependency.
⑧ Overloading not possible	Overloading possible. → function overloading → Operator "

CONCEPTS / FEATURES OF OOPS

ABSTRACTION - Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without representing the details.

Ex- A TV, which we can turn on and off, change the volume, channel, add external components such as speakers, VCRs, and DVDs players. But you do not know its internal details, i.e., how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.

DATA HIDING - is a software development technique specifically used in object-oriented prg. (OOP) to hide internal object details (data members).

OR

"Hiding the internal structure of object, protecting them for corruption."

→ Private and public are two types of protection available within a class.

Private members can be accessed by members of the class. Items marked with 'private' can only be accessed by methods defined as part of class.

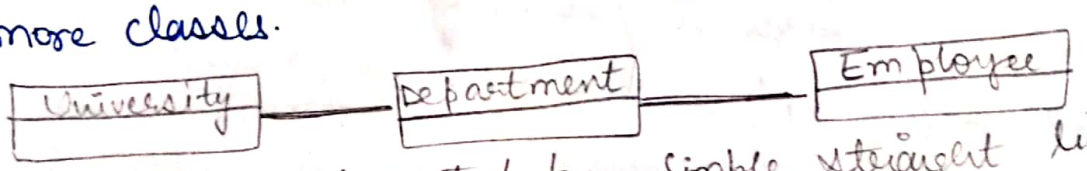
Public members can be accessed by members + objects of class. Public items can be accessed from anywhere in the program without restrictions.

ENCAPSULATION – is a process of combining data members and functions in a single unit called class.

→ It is also known as Information hiding concept.

→ The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

Association – An Association is the term used to represent the relationships among various objects of one or more classes.

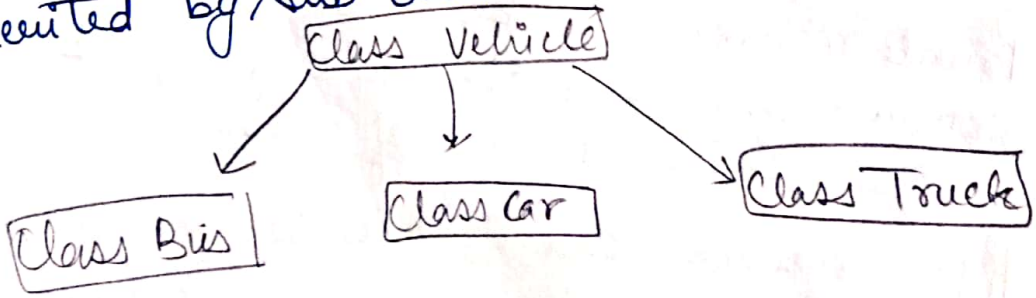


Association is represented by simple straight line and classes have been represented by rectangles

INHERITANCE – is a process in which one object acquires all the properties and behaviours of its parents objects automatically.

Derived class / Sub class :- The class that inherits properties from another class.

Base class / Super class :- The class whose properties are inherited by sub class.

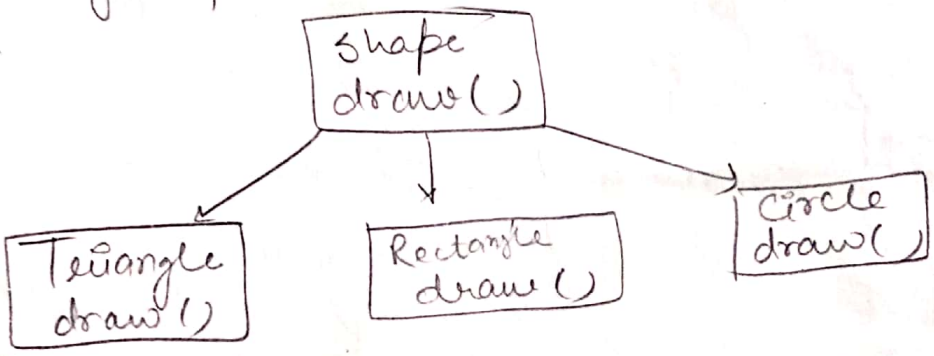


POLYMORPHISM – The word Polymorphism means having many forms.
'Poly' means 'many' & 'morph' means 'form'.

→ 2 Types of Polymorphism :-

① Compile time / Static binding Polymorphism – Function Overloading & Operator Overloading are perfect example of compile time polymorphism. (The process of making an operator to exhibit diff. behaviours in diff. instances

② Runtime / Dynamic Polymorphism – Function Overriding is an example of Runtime Poly.
↓
Using a single function name to perform diff. types of tasks



UNIT-2

BASICS OF C++

→ It is an OOP language. It was developed at AT & T Bell laboratories in the early 1979s by Bjarne Stroustrup.

→ Its initial name was "C with classes", but later on in 1983 it was renamed as C++.

→ C++ lang. is an extension of C lang. and supports classes, inheritance, function overloading & operator overloading which were not supported by C lang.

→ It is helpful to map the real-world problem properly

C++ Character Set

Letters (Alphabets) → A---Z, a---z

Digits → 0--9.

Special characters → +, -, *, /, (), {}, [], =, !, <>; ", $, %, #, <=, >=, @, ;, &, ?, _.

→ There are 62 letters and Digits character set in C++ (26 capital letters + 26 small letters + 10 digits).

Tokens: is a group of characters that are logically combine together. The programmer can write a program by using tokens.

- keywords
- operators
- Identifiers ← (Symbolic name) Identifier is a sequence of characters, taken from C++ character set.
- Literals
- Punctuators

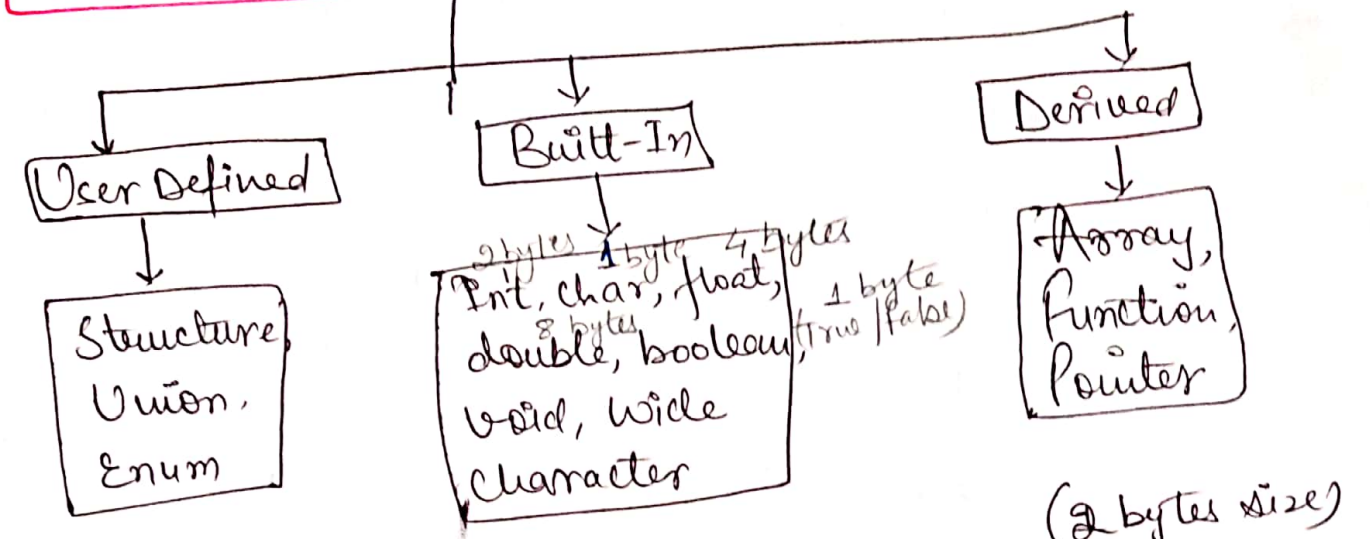
Keywords : Some reserved words in C++ which have predefined meaning to compiler.

→ Always written or typed in lower cases.

auto, break, else, default, private, public, class, char, if, int, goto, long, double, while etc.

Structure of a

DATA TYPES IN C++



① **Integer Data Type (int)** — Ranges (2 bytes size) − 32768 to +32768

- **Short** : −32768 to +32768
- **Long** : −2,147,483,648 to 2,147,483,648
- **Unsigned** : positive integers upto 65536.

② **Float** - has a decimal point. 27.4, -92.7 etc. (4 bytes size)

③ **Char** - (1 byte size) used to store character values in the identifier.

④ **Void** - Specifies the return type of a func^n. when the func^n is not returning any value.

→ Indicates an empty parameter list on a func^n when no parameters are passed.

→ A void pointer can be assigned a pointer value of any datatype.

(1)

VARIABLES - A variable is the most fundamental aspect of any computer language. It is a location in the computer memory which can store data and given a symbolic name.

→ The variables can be used to hold different values at different times during the execution of a program.

→ Before a variable is used in a program, it has to be defined so that the compiler make available the appropriate type of location in the memory.

→ The definition of a variable consists of a type name followed by the name of the variable.

Declaration of variable - When we use a variable in C++, we must first declare it specifying which data-type we want it to be. (like int, float, char etc) - followed by a valid variable identifier.

Ex - int a; float total; etc
char name;

If we want to declare more than one variable of same data type, we can declare all of them in a single statement by separating with commas. int a, b, c; Or int a;
int b;
int c;

Literals Or Constants - A number which does not change its value during execution of a program is called a constant or literals.

A keyword const is added to the declaration of an identifier to make it constant.

Ex - Const int rate = 50
Const char ch = 'A';
Const float Pi = 3.14;

② **Scope of Variables** A scope is a region of program.

A variable can be either of Global or local scope.

Global Variables: Global variables are defined outside of all the functions, usually on top of the program.

The global variables will hold their value throughout the lifetime of your program.

The global variable can be accessed by any function. Means global variable is available for use throughout your entire program after its declaration.

Local Variables: Variables that are declared inside a function or block are local variables

The scope of local variables is limited to the block enclosed in braces { } where they are declared means they can be used only by the statements that are inside that function.

Local variables are not known to functions outside their own. It is possible to have local variables with same name in different functions.

Ex.

```
# include <stdio.h>
int a,b ;                    /* global variables */
main ()
{
int c ;                      /* local variables */
---
---
}
```

outside the main block →

inside the main block

4) 3rd

Simple C++ Program

```
# include <iostream.h>
  using namespace std;
  int main()
  {
  cout << "Hello World!";
  return 0;
  }
```

//Section 1: Include Directive
//Section 2: Class Declaration and member.
//Section 3: Main function definition
//Section 4: Declaration of an object

Section-1 : The Include Directives

include <iostream.h> — ".h" contains c++ function declaration and macro definition

Lines beginning with hash sign (#) are directives for the preprocessors and called Header Files.

→ Header files contain definitions of Functions and Variables which is used into any c++ prog. by using the pre-processor # include statement.

Ex- # include <stdio.h> — used for get the input & output print the output.
include <conio.h> — when we use clrscr() in c++ prog.
include <iostream.h>
include <math.h> — contains declarations of mathematical functions in c++

This specific file (iostream) includes the declaration or information of the basic standard input-output library in c++ and it is included because its functionality is used later in the prog.

Section-2 Class Declaration & Member function

using namespace std;

The keyword Using technically means, use-this whenever you can. In this case, to the std namespace.

std is the abbreviation of standard. std is the standard namespace. cout, cin and a lot of other things are defined in it. Also call these funct. using std::cout, std::cin etc.

→ All the elements of the standard ANSI c++ library we declared within namespace std.

Section 3 : Main function definition

main() function is the entry point of any c++ program. It is the point at which execution of program is started. When a c++ prog. is executed, the execution control goes directly to the main() func. Every c++ prog. have a main() function.

ex.
```
Void main()
{
---
---
}
```

Void : Void is a keyword in c++, Void means nothing, whenever we use Void as a function return type then that func. nothing return. Here, main() func. no return value.

int : In place of void we can also use **int** return type of main() funct., at that time main() return integer type value.

main : is a name of function which is predefined function in c++ library.

Section 4 : Declaration of an object

Right after parantheses { } we can find the body of the main_func. enclosed in braces {}. cout << "Hello World!"; This statement is used to display output on the screen of the computer. Cout is the name of the standard output stream in c++, meaning is to insert a sequence of characters.

Cout is declared in the iostream standard file to declare (inform) that we are going to use this specific namespace earlier in the code. Statement always ends with a semicolon (;).

Return 0; This statement causes the main function to finish

FUNCTIONS IN C++ (Function Overloading)

→ Functions helps in dividing program/code into smaller func^n. And Increases reusability of code.

→ Makes it possible to reduce size of a program by calling and using them at different place.

Main() func^n in C++

→ starting point

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

Function Prototyping (Declaration) – Prototype describes the func^n interface to the compiler by giving details such as no. and type of arguments and type of return values.

template

```
void add (int i, int j);
```

Function Declaration

(void means such that return expect no return value)

return type | func^n name | arguments

When a function is called, compiler uses the templates to ensure that proper arguments are passed & return value is correct.

Example, If we call `add(1,2);` then compiler check this call with the template, whether the func^n name is correct or not and also check the no. of arguments & type of arguments passed.

Here, `add(1,2);` is correct

`add('a',2);` is not correct and result is compilation error. Because the argument type is 'char' and the function declared has integer type argument

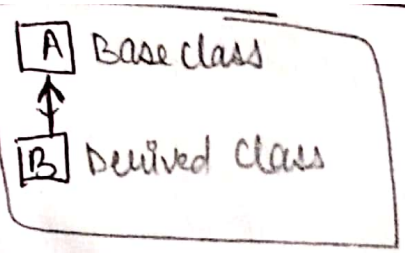
```
#include <iostream>
int main()
{
  int i,j;
  void addnum(int a, int b);  ] function declaration
  cin >> i;
  cin >> j;
  addnum(i, j);  if this is correct
  return 0;          | then execute
}
void addnum (int a, int b)
{ cout << "The sum is : "<<a+b<<"\n";
}
```

name

check

Containers
Exceptions
Templates
File ptes
I/O streams

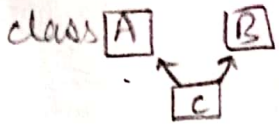
Scanned by CamScanner




```
Class A
{
  ---  ← members of class A
};
Class B : public A
{
  ---  ← members of class B
};
```

Template

(2) MULTIPLE INHERITANCE (If a class is derived from more than 1 base class)

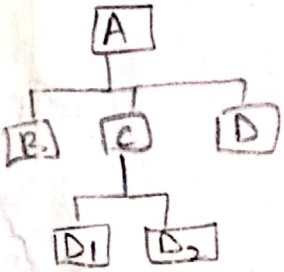



```
Class A
{---
};
class B
{---
};
Class C : Public A , Public B
{---
};
```

(3) MULTI-LEVEL INHERITANCE (If a class C is derived from class B which is derived from class A and so on).

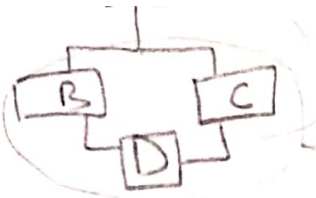

```
Class A
{---
};
Class B : public A
{---
};
class C : public B
{---
};
```

(4) HIERARCHICAL INHERITANCE (Several class are derived from single base class.)



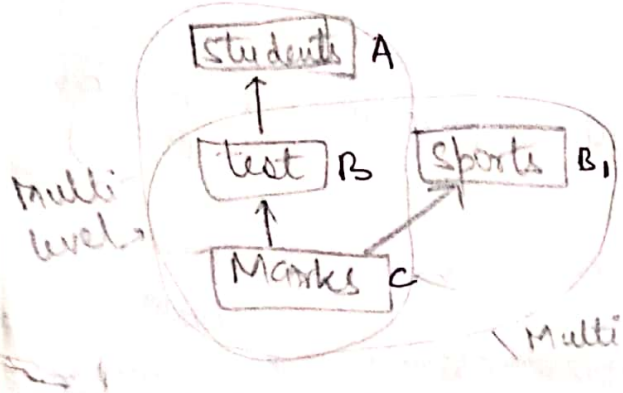

```
Class A
{---
};
Class B : public A
{---
};
Class C : public A
{---
};
Class D : public A
{---
};
Class D1 : public C
{---};
```

```
Class D2 : public C
{---
};
```



Hierar

Multiple




```
Class A
{ --
};
Class B : public A
{ --
};
Class C : public B, public B1
{ --
};
```

FUNCTION OVERLOADING.

The process of using two or more functions with ~~same name~~ same name but differing in the signature is called func.ⁿ overloading.

Or

Using same function name to create functions that performs variety of different tasks.

→ Same functions name but different argument.

→ Overloading of functions with diff. return types are not allowed.

eg. `int add(int a, int b);` | `int add(int a);`

`add(5,2)` | `add(5)` → [Compiler checks no. of arguments passed ~~on the basis of~~ and argument data types, then decide which funⁿ will be called]

Function Selection Steps :

1. Compiler first tries to find exact match.
2. If exact match not found, compiler uses integral promotions to actual arguments. [char → int, short → int, float → double]
3. Then compiler uses built-in conversi[illegible] actual argu[illegible]. But in case of multiple match there is ERROR.

long ← function
int
double ← function
} Ambiguity, it will give error.

Operator Overloading

The operator overloading is a concept in which we can give special meaning to an existing operator..

→ It provides a flexible option for creating of new definition for most of the c++ operators.

→ Operator overloading enables us to make the standard operators, like +, −, * etc to work with the objects of our own data types.

add(int) 5+3 = 8 numeric value

add(float) a+b = ab Concatination

Operators that cannot be Overloaded

1. Membership operator (.) member selection
2. Scope resolution operator (::) used in defining func^n. [uses names instead of operator]
3. Size of operator (sizeof) So we can't pass any name to any function
4. conditional operator (?:)
5. pointer to member operator (.*, →*) used in structures

Syntax :

```
return_type class_name :: operator op (op argument list)
{
   ---  Body of function
}
```

↑ outside the class

```
return type Operator op ( )
```

Operator → function; op → symbol of operator

← Inline definition

Overloading Unary operators

Unary minus (Unary −)

5 ⇒ −5

Suppose a class space s ← object

(−s)

data members of class { −x, −y, −z } ⇒ −−x = +x

if (s) then only −x, −y, −z ⇒ −x, −y, −z

```
class space
{ int x;
  int y;
  int z;
public:
 space()
  { }
 space (int x1, int y1, int z1)
  { x = x1;
    y = y1;
    z = z1;
  }
void operator -()
  { x = -x;
    y = -y;
    z = -z;
  }
 void show()
 {
 cout << "\n x is = " << x;
 cout << "\n y is = " << y;
 cout << "\n z is = " << z;
 }
};  void main()
 { space s(10, 20, -30);
   s.show();
   -s;
 cout << " after applying new definition";
   s.show();
 }
```

```
returntype operator op()
{
}
```

```
10  20  -30
-10 -20 +30
```