History of C++ Language

Developed By : Bjarne Stroustrup

Company :- AT&T (American Telephone and Telegraph) in Bell Labs USA.

Launching :- 1979, 1983

Initially in 1979 C++ was known as "C with Classes",

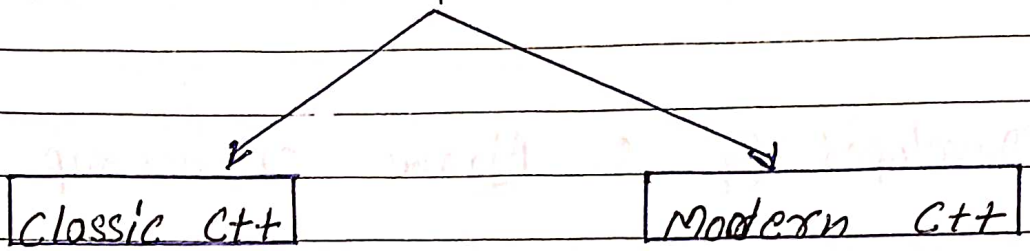
In 1983, the name of the language got modified from "C with classes" to C++.

★ Why C++ is called as C++?

Ans:- The name C++ justifies that this language contains/ supports all the primary topics of C language like Data types, Operators, Control statements, Functions etc. In addition we get something more, something extra and that extra part is "CONCEPTS OF OBJECT ORIENTED PROGRAMMING".

Date
Page

★ Types of C++ / Flavors of C++ :-



- Classic C++ is only supported by Turbo C compiler.
- Modern C++ is supported by All new compilers like :- Gcc, IBM C++, Intel C++, Visual C++, Clang, LLVM.

Modern C++ :-

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2020 as C++ 20

Version :-

C++ 11	2011
C++ 14	2014
C++ 17	2017
C++ 20	2020

Date
Page

Two Important difference :-

Classic C++	Modern C++
• Support .h extension with C++ header file like - iostream.h, fstream.h.	Prohibits the .h extension with C++ heade file like - iostream, fstrem.
• No concept of namespace.	Heavily use concept of namespace.

Initially we will start with Classic C++ (till Constructors) and Afterwards we will migrate to modern C++.

Compiler And IDE's used

Classic C++ :-

Compilers	IDE
TC	Turbo C++

Modern C++ :-

IDE	Compilers
• Code Blocks	Gcc
• Dev C++	Gcc
• visual studio	Microsoft C++
• C - Free	C - Lang
• XCode	LLVM / Low Level Virtual Machine

Date
Page

•	Eclipse	GCC

★ Difference in C & C++ (Syntax)

	C	C++
1)	The header file which is most common & popular in C lang. program is "stdio.h", as it provides us Declaration / PROTOTYPE of 2 very Important functions called as printf() and scanf() used for console output and console input respectivly.	• The header file which is most common and popular on C++ (classic as well as modern) is iostream.h, as it provides 4 predefined objects called cout, cin, cerr, and clog. • In these objects the most popular are cout and cin objects used for displaying text on console as well as reading input.
2)	printf("Good Morning");	cout<<"Good Morning";
3)	int a=10; printf("%d", a);	int a=10; cout<<a; → Insertion operator or Inserter
	Char b='x'; printf("%c", b);	Char b='x'; cout<<b;
	float c=1.7; printf("%f", c);	float c=1.7; cout<<c;

This statement is using CASCADING of INSERTION operator

4)
```
int   a = 10;
char  b = 'x';
float c = 1.7;
printf("%d %c %f");
```

```
int   a = 10;
char  b = 'x';
float c = 1.7;
cout<<a<<b<<c;
```

5)
```
int a;
Scanf("%d", &a);
```

```
int a;
cin>>a;
```

6)
```
int a;
char b;
float c;
scanf("%d %c %f", &a, &b, &c);
```

```
int a;
char b;
float c;
cin>>a>>b>>c;
```

Extraction operator or Extractor

7)
```
printf("Good in Morning");
```

```
cout<<"Good in Morning";
or
cout<<"Good<<endl<<Morning";
```

manipulator

8)
```
int i;
for(i=1; i<=10; i++)
  {
    printf("\n%d", i);
  }
```

This code will have different Behaviour in classic c++ and in Modern c++.

```
int i;
for(i=1; i<=10; i++)
  {
    cout<<i<<endl;
  }
```

or

```
for(int i=1; i<=10; i++)
  { cout<<i<<endl;
  }
```

Date
Page

- In classic C++ it will 1--11 ;

- In Modern C++ it will give Error because in Modern C++, these variables which Are declared in the Initialization section of the for loop get destroyed at the loop is over.

9	void main()
	{
	- - - -
	}
	
		int main()
		{
		- - - -
		- - - -
		return 0
		}

A Sample program in C, Classic C++ and Modern C++ :

In C Language :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a, b, c;
    clrscr();
    printf("Enter 2 int : ");
    scanf("%d %d", &a, &b);
    c = a+b;
    printf("Sum is : %d", c);
    getch();
}
```

In classic C++ :—

```
#include<iostream.h>
#include<conio.h>
int main()
{
    clrscr();
    int a,b;
    cout<<"Enter 2 int:";
    cin>>a>>b;
    int c;
    c=a+b;
    cout<<"sum is:"<<c;
    getch();
    return 0;
    }
```

In Modern C++ :—

```
# include<iostream>
using namespace std;
int main()
{
   int a,b;
   cout<<"Enter 2 int ";
   cin>>a>>b;
   int c;
   c = a+b;
   cout<<"Sum is "<<c;
   return 0;
}
```

" Introduction To OOP "

Q) What is object oriented programming ?

- OOP is a programming methodology and is supported by maximum programming language like C++, Python, Java, Javascript, PHP, C# etc.
- The world in which we live is full objects, Every living or non-living thing is nothing but an object.
- Thus OOP says we should develop our program keeping objects as the center point.
- Data security is given much more importance.

Q) What is procedure oriented Programming ?

POP is a programming methodology Supported by C programming language.

The word proccdure means FUNCTION And so the term procedure oriented programming means FUNCTION ORIENTED PROGRAMMING.

Date
Page

In other words it means that we should break our program into multiple Functions and define each function independently and call it from other parts of the program as and when the requirement arises.

★ Advantage of POP

1) It's easy to implement

2) The ability to re-use the same code at different places in the program without copying it.

3) An easier way to keep track of program flow for small codes

4) Needs less memory.

★ Disadvantage of POP.

1) very difficult to relate with real world objects.

2) Data is exposed to whole program, so no security for data.

3) Difficult to create new data types

4) Importance is given to the operation on data rather than the data.

Q) what is an object in OOP. ?

Ans:- In programming any real world entity which has specific attributes or features can be represented as an object.

In simple words, and object is something that possess some characteristics and can perform certain functions.

Q) what does an object Contain ?

Every objects is composed of just 2 things :

i) Data / Information : represents the values objects contain.

ii) Function / Actions : represents the behaviours of an object can perform.

Date
Page

For example, car is an object and can perform function like start, stop, drive and brake.

These are the functions or behaviours of a car.

And the characteristics or attributes are color of car, mileage, maximum speed, model, year etc.

☆ Classes :-

Now to create / represent objects we first have to write all their attributes and behaviours under a single group.

This group is called a class.

Thus a class is an architecture / blueprint of the object. It is a proper description of the attributes and methods of the objects.

For Examples :- The design a car of same type is a class. we can create many objects from a class. Just like we can make many cars of the same type from a design of car.

Date
Page

★ General Syntax of Classes :-

1)

```
Class <class_name>
{
    // data members

    Public :

    // member functions

};
```

2)

★ Structure in C & Class in C++

Structure "C"	Structure "C++"

Struct student
{
int roll;
char grade;
float per;
};
void main()
{
Struct student s;
s.roll = 10;
--- 	
--- 	
}
 	
}
	class student
	{
	int roll;
	char grade;
	float gper;
	} Data Members
	
	void get();
	void show();
	} member function
	};
	int main()
	{ student s; object
	s.roll = 10;
	--- 	
	}
	
	}

Date
Page

Q) What are Private And Public in C++ ?

Ans :- Private and Public are keywords :

- These keywords are popularly called as access specifiers or visibility modes.
- They control that what members of the class will be ~~ass~~ accessible from outside the class and what members could not be accessed from outside the class.
- The members which are declared as public can be accessed from outside the class, but the members which are declared as private can never be accessed from outside the class.
- Normally we always declare data members of the class as private while member function as public.
- In C++, whenever we design a class by default everything is private, so there is no need to mention ~~to~~ the keyword private. But to make members public we will have use the key word public.

Date
Page

There is one more keyword in C++ which is also called as access specifier and this keyword is protected. But discussion about it will only make sense when we discuss the chapter INHERITANCE.

☆ Developing the first OOP Program :-

```
#include<iostream.h>
#include<conio.h>
class student          <- Name of the class
{
  private:             <- Access specifiers and since it is the default Acess specifier so we can avoidit
    int roll;
    char grade;        } private Data members OR private Instance variables
    float per;
  public:              <- Access specifier and this needs to be mentioned so that we can call these member functions from outside the class.
    void get();
    void show();       } public member Function.
};
```

Daclaration of the class

Date
Page

★ Syntax of defining member function :—

```
<return type> <class_name> :: <mem_fn_name>(<arg>)
{
        // body of the member function
}
```

Syntax of calling a member function :—

```
<obj_name><mem_fn_name>();
```

★ First Program In OOP

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

class student
{
    private :
        int roll;
        char grade;
        float per;
    public :
        void get();
        void show();
};
void student :: get()
{
  cout<<"Enter roll, grade and per";
  cin>>roll>>grade>>per;
}
```

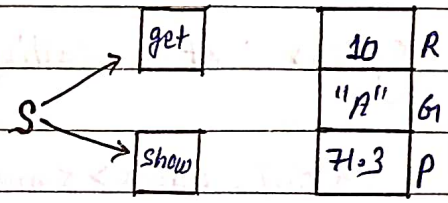
private Data member

public Data member.

Date
Page

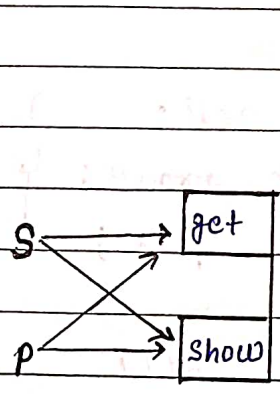
Scope Resolution operator

```
void student :: show();
{
    cout<<" Roll = "<<roll<<endl;
    cout<<" Grade = "<<grade<<endl;
    cout<<" per = "<<per<<endl;
}
int main()
{
    clrscr()
    student S;
    S.get();
    S.show();
    getch();
    return 0;
}
```



when we create 2 objects of same class :

```
int main()
{
    clrscr();
    student S P;
    S.get();
    S.show();
    P.get();
    P.show();
    getch();
    return 0;
}
```



10	R
'A'	G
71.3	P

	R
	G
	P

Q) WAP to calculate and print the sum of two integers given by the user :

```
#include<iostream.h>
#include<conio.h>
class sum
{
    int a;
    int b;
    int c;

    public:
        void get();
        void add();
        void show();
};

void Sum :: get()
{
    cout<<" Enter 2 int : ";
    cin>>a>>b;
}
void Sum :: ~~show~~add()
{
    c = a+b;
}
void Sum :: show()
{
    cout<<" Numbers are "<<a<<" and "<<b<<endl;
    cout<<" Their sum is "<<c;
}
```

Date
Page

OUTPUT

```
int main()
{
    clrscr();
    Sum obj;
    obj.get();
    obj.add();
    obj.show();

    getch()
    return 0;
}
```

```
Enter 2 int : 10 20
Numbers are 10 and 20
Their sum is 30
```

☆ Relation Between OOP & C++

	OOP	C++
i)	It is a Methodology or approach of writing/ developing programs.	i) it is a lang. which is based on upon the principles of OOP.
ii)	It teaches us those principle or concepts using which we can develop program which are SECURE as well as can MODEL REAL WORLD SITUATION.	ii) in other words C++ provide us syntaxes to implement the concepts we study in OOP
iii)	It came in the year 1967 and world's first object oriented lang. was SIMULA 67.	iii) for ex, OOP says we should have secure data i.e. we should protect our data from illega and unauthorized access and to achieve this we use the keyword private given by C++.

iv) Simula language was used in labs to experiment simulation.	

Types of Member functions :-

- Ass Accessors :-

These are those member function which NEVER change the value of the data members of the calling object. In other words they only Access the values of data members but they do not change them.

- Mutators :-

These are those member function which change / manipulate the values of the data members of the calling object.

★ Terminologies Use in C++ & OOP

	C++ words	OOP words
1)	Objects	1) Instance, Entity.
2)	Data members	2) Attributes, properties, features, Fields

3)	Member Functions	3) Method, Actions, Behaviou
4)	Member Function call	Message Passing.

↓

It is the communication Between an INSTANCE and It's BEHAVIOUR.

☆ Classification of Programming Languages :-

According to OOP :-

According to OOP, all the programming language of the world can be categorized in to 4 categories and this categorization is done on the basis of support a language provide to, three MOST IMPORTANT principles of OOP called as :

a) Encapsulation
b) Polymorphism.
c) Inheritance

Following are the details of this categorization :

Date
Page

1) NON Object oriented Languages :

These are those language which do not at all support any kind of the three principles mentioned above.

Ex :- C, Cobol, Pascal, Fortran etc.

2) Object Based Languages / Partially object oriented Language :-

These are those language which do not support all the principles mentioned above but they support same of them. The most popular ex. in this category is the language called JAVA SCRIPT, It supports Encapsulation, Polymorphsim, but it does not support ~~the~~ inheritance.

Ex :- VB Script, ActionScript JBScript etc.

3) Object oriented Language :-

These are those language, which are atleast support all the three principles mentioned above but they never force a programmer to always use these principles in his program.

Ex :- Classic C++, modern C++, Python, PHP etc

4) Pure object oriented / Strict object oriented / Full object Oriented Languages :—

These are those languages which not only support the above three principles but they perform all of their activities in an OBJECT ORIENTED WAY.

Their three important characteristics are :

a) No global declaration.
b) It is compulsory to use "class" in even the simplest program of these language.
c) They don't support variables as everything in them is object.

Strictly Speeking, them in today's world a language which can be called 100% pure object oriented (without any controversy) is Smalltalk.

★ What about JAVA, C#, Scala etc?

These language can also be turned as pure object oriented languages but some people think that they are not pure object oriented as they

Support concept of data types and variables. But ~~thee~~ technically speaking, language like Java allow us to convert even variable into objects using the concepts of wrapper classes. So if this point is taken into consideration then even Java can be called as a pure object oriented language.

★ "Three Pillars of OOP"

- Such OOP is collection of many principles but amongest them are three most important principles called as Encapsulation, Polymorphism and Inheritance.

Encapsulation :—

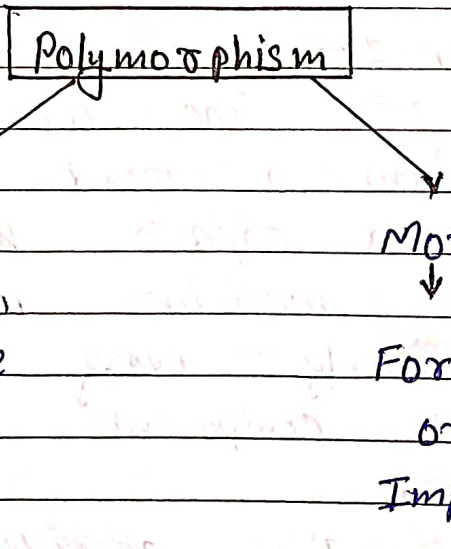
The word Encapsulation is derived from a word called "capsule" Just like a capsule has a collection of multiple medicines wrapped in a single block, similarly every program has only 2 important components which are data & function.

Thus according to OOP these data & functions must always be kept together. So in terms of OOP, we

Can Say that the word Encapsulation is defined as "Bundling / wrapping up of data and functions action on those data withing one single unit.

Now Since a class allows us to declare both data & function in it's body, we can Say that it is an example of or implementation of the principle of Encapsulation.

The most important benefit offered by Encapsulation is Data Security i.e. data members of a class are kept as private and thus they can not be accessed directly from non member functions. This keeps the data safe and secure from any kind of illegal and unauthorized access as well as makes it secure.



Date
Page

- The word Polymorphism is derived from a combination of 2 words which are called Poly and Morph. The word Poly means multiple and the word Morph means forms Thus the word Polymorphism. means having Multiple forms.

- In other words we say that if a single entity can behave differently in different situation, then it is behaving Polymorphically.

- For ex:- in c++, if a programmer wants then he can redefine the built in operator + in such a way that this operator can be used to concatenate 2 strings, just like it.

In C++ Polymorphism is divided in 2 parts:

a) Compile Time Polymorphism: Polymorphism that is resolved during compiler time is known as compile Time polymorphism.

Ex:— Function overloading, Operator overloading

b) Run time polymorphism :- Run time polymorphi is a process in which a call to an overridden method is resolved at run time, thats why it is called runtime polymorphism.

Ex:- Virtual function, Pure virtual Function and Abstract classes.

"INHEITANCE"

- To inherit means to acquire features of an existing entity in a newly created entity.
- Just like a child inherits the features of his or her parents, similarly in OOP we can inherit the feature (data & member functions) of an existing class in a newly created class.
- When we do this we say that we have implemented inheritance.
- The class which gets inherited is called as parent class (OOP term) or base class.

Date
Page

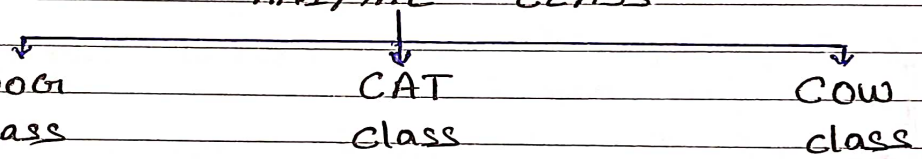
- The class which inherits is called as child class (OOP term) or derived class (C++ term) or sub class (python or java term)

- Thus via inheritance the programmer of derived class gets the advantage of accessing members of the base class also using the object of derived class.

- The most important benefit offered by inheritance is code reusability.

- This means that the derived class programmer is not required to

Ex :—



Creating Parameterized Member Functions.

1) Takes something And Returns Something.
2) Takes Something And Returns Nothing.
3) Takes Nothing And Returns Something.
4) Takes Nothing And Returns Nothing.

A class can have all 4 types of member functions depending on the requirements.

In today's session we are going to learn how to pass arguments to member Function ?

```
#include <iostream.h>
#include <conio.h>
#include <string.h>

Class Emp. {
    Int age ;
    Char name [20];
    float Sal ;
 Public :
        void set ( int , char* , float );
        void show();
};
void Emp :: set (int a , char *p , float s)
{
    age = a;
    strcpy (name, p);
    Sal = s;
}
```

```
void Emp :: Show()
{
   cout<< age << ", " << name << ", " << sal << endl;
}
int main()
{
    Emp E, F;
    E.set(25, "Rahul", 30000.0);
    F.set(28, "Chetan", 25000.0);
    E.show();
    F.show();
    getch();
    return 0;
}
```

How many ways are there in C++ to initialize data members of the class ?

1.) Using Cin (user input) : This will be done at runtime and user will give the values. Use this when you want to initialize every object with diff values and that to from the user.

2.) Using Constant Initialization : The programmer will directly assign the values through member function.

Use this when we have some data for which every object will have same.

3.) Using parameterized member function : use this when you want to pass the values from the main() function.

4.) use in class initialization : But it only work with. Modern C++.

Q5 Write an object oriented program to create a class Factorial which should calculate and print the factorial of the number given by the user. Make sure that your code contains.

```
#Include <iostream.h>
#include <conio.h>

class Factorial
{
    int n;
    int f;
Public :
    void init();
    void get();
    void calculate();
    void show();
};
```

```
Void Factorial :: init()
{
    f = 1;
}
Void Factorial :: get();
{
   cout<<" enter no: ";
   cin>>n;
}
void Factorial :: Calculate ()
{
    for (int i=1 ; i<=n ; i++)
      {
         f = f*i;
      }
}
void Factorial :: show()
{
   cout<<" No is "<<n<<endl;
   cout<<" Fact is "<<f;
}
int main()
{
    factorial obj;
    obj. init();
    obj. get();
    obj. calculate();
    obj. show();
    getch();
    return 0;
}
```

Assignments :-

WAP to create a class called Date having 3 integer data members called day, month and year.
Provide following 2 member functions in your class :

1) setdata(); this member function should accept 3 arguments and intialize all the data members.

2) Showdata(); this will be non parameterized function and it will display the date values.

Now. create the fuction main(), declare 2 objects of Date class and initialize the first. your birthdate and initialize the second object with todays date.

Finally display both the dates.

Date
Page

"Constructor"

What is Constructor ?

In C++ (as well as Java also) a Constructor is a Special member fuction of the class having the following important properties:

1) They have the same name as that of the class.
2) They do not have any return type not even void.
3) They are Automatically called by the C++ Compiler as soon as ~~of~~ the object of a class gets called.

Previous program using constructor :

```
#include <iostream.h>
#include <conio.h>
Class Factorial
{
  int n;
  int f;
  public :
        Factorial ();
        void init();
        void get();
        void calculate();
        void show();
};
```

```
void Factorial :: Factorial()
{
    f = 1 ;
}
void Factorial :: get()
{
    cout<<" Enter no:";
    cin>>n;
}
void Factorial :: calculate()
{
    for (int i=1 ; i<=n ; i++) {
        f = f* i ;
    }
}
void Factorial :: show()
{
    cout<<" No. is "<<n<<endl;
    cout<<" Fact is : "<<f;
}
int main()
{
 Factorial obj;
 obj.get();
 obj.calculate();
 obj.show();
 getch();
 return 0;
}
```

Date
Page

The Default Constructor :—

1) In c++, there is a special rule regarding constructors.

2) The rule is that if a programmer will not define any constructor in his class, then the c++ compiler will automatically add a special constructors in the class which is called as the Default Constructor.

3) For ex. If the name of the class is Factorial, then the compiler generated default constructor of the class will be :

```
Factorial :: Factorial()
{
}
```

4) We must remember that default constructor is supplied by the compiler only when we have not created any constructor ourself. Otherwise the compiler will withdraw the default constructor.

5) Default constructor does not take any argument. Its has no parameters.

Date
Page

☆ Creating Parameterized constructor :-

Parameterized Constructor :-

1) In C++, just like we can have parameterized member functions, similarly we also can have parameterized constructors.

2) But, if we create a parameterized constructor in our class, then each and every object of the class that we are going to create must compulsory be parameterized.

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class Emp
{
   int age;
   char name[20];
   float sal;

 public :
        Emp(int, char*, float);
        void show();
};
Emp :: Emp (int a, char *p, float s)
{
        age = a;
        strcpy(name, p);
        sal = s;
}
```

```
void Emp :: show()
{
  cout << age << ", " << name << ", " << sal << endl;
}
int main()
{
  Emp E(25, "Amit", 30000.0);
  Emp F(28, "Abhay", 29000.0);
  E.show();
  F.show();
  getch();
  return 0;
}
```

Output :-

25 Amit 30000

28 Abhay 29000

"Function Overloading"

- In C++, if within the same Scope (class or program) we have declared 2 or more functions with the same name, then we say that it is Function Overloading.

But if we are declaring two or more functions with the same name, then we must provide some difference in the prototype of these functions and this difference is in items of the Arguments.

The difference of Arguments can be of 3 types :

1) Difference in number of arguments :

for ex :- void volume(int);
void volume(int, int, int);

2) Difference in data types of arguments.

for ex :- void area(int);
void float(float);

3) Difference in order of arguments

for ex :- void show(int, float);
void show(float, int);

Special Point :-

- we can never upload 2 or more functions just on the basis of their return type.

- This means, that overloaded functions compulsorily must differ with each other with respect to their arguments and if

they only differ in items of their return types then the code will not even compile

- So following Declaration will give Syntax error :

```
void show();
int show();
float show();
```
} ✗ Syntax error.

→ Why we can't overload functions just on the basis of their return types :

Exe→
```
int show();
void show();
int show();
    main
{
    int x;
    x = show(); // Perfectly okay.
    show(); // confused ?
}
```

This is because, it is not compulsorisory To receive or use the value returned by a function.

So the compiler will not be able to Determine, that whether the call is being made to a function with void Return type or to any other function whose return type is not void, but we are not receiving its return values.

Ex :-

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

void volume(int);
void volume(int, int, int);

int main()
{
    int choice;
    clrscr();
    cout<<"select a figure :";
    cout<<endl<<" 1. cube "<<endl<<" 2. cuboid : ";
    cin>>choice;
    switch(choice)
    {
        case 1 :
                int s;
                cout<<"Enter side of the cube :";
                cin>>s;
                volume(s);
                break;
        case 2 :
                int l, b, h;
                cout<<"Enter l, b, h of cuboid :";
                cin>>l>>b>>h;
                volume(l, b, h);
                break;
        default :
                cout<<" wrong choice ";
    }
    getch();
    return 0;
}
```

```
void volume (int s) {
    cout<<" vol of cube is "<<s*s*s<<endl;
}
void volume (int l, int b, int h) {
   cout<<" volume of cuboid is "<< l*b*h <<endl;
}
```

Output :-

Select a figure :

1. Cube
2. Cuboid

1

Enter side of the cube : 3

vol of cube is 27

What is the benifits of overloading :

1) The overhead of remembering function names does not comes on the programmer calling the function. He can simply remember just one name and using that he can call different versions of the functions

2) The code becomes more Symmetrical as well as clean if for similar task we use functions with same names.

Date
Page

Does overloading really exist ?

- Surprisingly, the ~~answers~~ answer to this question is both **yes** and **no**.

- But to understand this, we should first recall the way a program is compiled and executed.

- We know that the code which we write is called as source code and this source code is then converted to machine code by the c++ compiler.

- But we should also remember that compilers never run any program. They just convert it to machine code and send it for execution to the OS. But every OS has a restriction which is that all functions in the machine code version must be uniquely named.

- Thus while converting our source code to the machine code, the c++ compiler converts all the overloaded function names to same unique name.

- Specially for this purpose, it uses a software built into c++ compiler called as name Mangler.

- Thus when the machine code of our program gets generated then each and every function becomes uniquely named and overloading gets totally removed.

- Hence at the source code level overloading does exists, but at the machine code level no overloading, So the answer to the above question is both yes and no.

☆ "Constructor Overloading"

Just like we can overload functions, we also can overload Constructors in our class. This means, That in a single class we can have multiple constructors. But if we overload constructors then we must ~~remb~~ remember that every constructor must appear different from other constructors with respect to its arguments and as before this difference can be of 3 types:

1) Number of Arguments
2) Data type of Arguments
3) order of Arguments.

Date
Page

- Now when we create object of our class, then the compiler will select the that constructor which closely maches the arguments passed while creating the object.

Q) WAP to create a class called Box having 3 integer data member for storing l, b and h.

Provide following member functions/ constructors in your class:

1) A non parameterized constructor which should accept values from user and then initialize the object.

2) A single parameterized constructor which should accept an int as argument and intialize all data members with it. i.e. like a cube.

3) A triple parameterized constructor to accept 3 integer values as arguments and initialize the object members with them i.e. like a Cuboid.

4) A member function called showV which should display the values of all the data members.

Finally design the function main(), create three objects of Box class in such a way that each object called a different constructor. Then I display their values.

Program :—

```
#include <iostream.h>
#include <conio.h>
Class Box
{
    int l, b, h;
    public :
        Box();
        Box (int);
        Box (int,int, int);
        void show();
};
Box :: Box ()
{
    cout<< " Enter l, b, h : ";
    cin>> l >> b >> h;
}
Box :: Box (int s)
{
    l = b = h = s;
}
Box :: Box (int i, int j , int k)
{
    l = i
    b = j
    h = k;      }
```

```
void Box :: show()
{
    cout<<l<<", "<<b<<", "<<h<<endl;
}
int main()
{
  Box B1;
  Box B2(10);
  Box B3(5,7,9);
  B1.show();
  B2.show();
  return 0;
}
```

- A copy constructor is a special constructor of our class.
- A copy constructor is used for copying one object to another
- It is called whenever we create a new object and want to initialise it with the values of an existing object.

How do we create a copy constructor?

To create a copy constructor in our class we have to declare a constructor which should accept the reference of the object

of it's own class of arguments.

Reference Variable :--

- Reference variable in C++ are a mechanism using which a programmer can refer to a single memory location, using multiple different names.
- In other words reference variables allows us to create aliases for existing variables.
- For a programmer a reference is considered to be an easier alternate to pointer.
- This is because like pointer, a reference operates on other variables. But pointers have a very typical syntax because they make use of indirection operator i.e. on the other hand reference variable does not make use of any indirection operator and allows us to directly access / change the variables values to which it is refering.

Do reference variable really share the address of the variable they refer?

Date
Page

- No, a reference is internally a pointer. Thus it has a separate memory location of it's own. But the compiler makes the programmer believe that reference variable and the variables to which they are refering have the same address.

- This is because whenever we use the name of a reference variable, the compiler simply prefix it ~~which~~ with a '*'. So cout<<p; becomes cout<<*p; and cout<<&p; becomes cout<<&*p;

Syntax of Reference Variable :

<data type> & <ref_var> = <var_name;

Ex:-

1) int a = 10;
   int &p = a; // initialization must be done along with declaration

2) int a = 10
   int &p; // ERROR
   p = a;

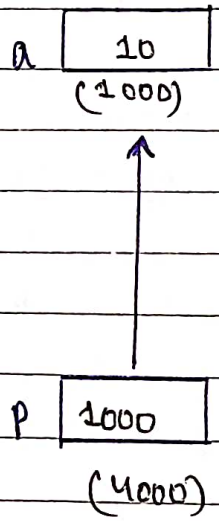
3) int a = 10;
int &p = a;
cout<<p; // They have a simple syntax as compared to pointer.
p++;
cout<<a<<","<<p;

Output :— 10
11, 11

Comparison between Pointer and Reference variable :

- Pointer :

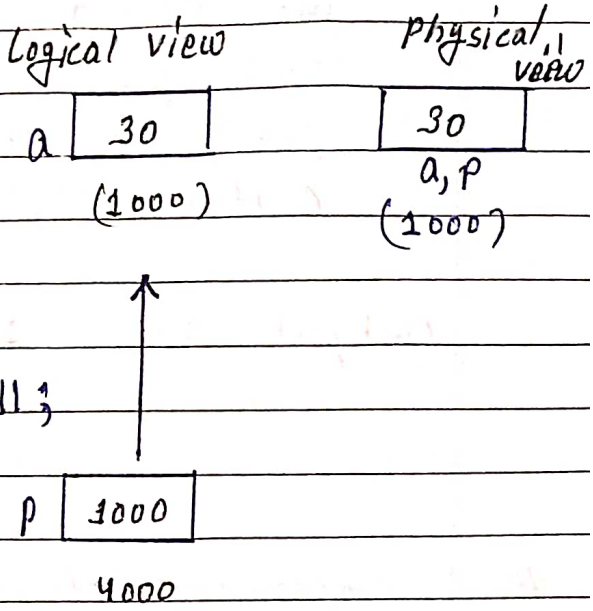
```
int main()
{
    int a = 10;
    int *p;
    p = &a;
    cout<<*p;
    getch();
    return 0;
}
```



Date
Page

Reference Variable :

```
int main()
{
    int a = 10;
    int &p = a;
    a = 30;
    cout<<a<<", "<<p<<endl;
    getch();
    return 0;
}
```



Pointer V/s Reference Variable :-

	Pointer	Reference Variable

1)	int a = 10;
	int *p = &a; // ok
	OR
	int a = 10;
	int *p;
	p = &a; // ok
		1) int a = 10;
		int &p = a; // ok
		OR
		int a = 10;
		int &p; // ERROR
		p = a;

2)	int a=10, b = 50;
	int *p = &a;
	cout<<a<<". "<<*p<<", "<<b;
	p = &b;
	cout<<a<<", "<<*p<<", "b;
		2) int a=10, b=50;
		int &p = a;
		cout<<a<<". "<<*p<<", "<<b;
		p = b;
		cout<<a<<", "<<*p<<", "b<<endl;

Output :

10, 10, 50

10, 50, 50

2)
```
int arr[5] = {10,20,30,40,50};
int *p = arr;
int i;
for(i=0; i<5 ; i++) {
    cout<< *p<< endl;
    p++; }
```

Output :
10
20
30
40
50

4)
```
int a=10,b=20, c = 30;
int *p[3] = {&a, &b, &c };
int i;
for (i=0; i<3 ; i++ ) {
    cout<< *p[i] <<endl;
    p++ ; }
```

Output :

10

20

30

Output :

10, 10, 50

50, 50, 50

3)
```
int arr[5] = {10, 20, 30, 40, 50}
int &p = arr[0];
int i;
for(i=0; i<5 ; i++) {
    cout<<p<< endl;
    p++ ;    }
```

Output :
10
11
12
13
14

4)
```
int a=10, b= 20 , c =30;
int *p[3];    // ERROR
```

5)
```
int a=10;
int *p;
int **q;
p = &a;
q = &p;
cout<<a<<", "<<*p<<", "<<**q;
```

Output : 10, 10, 10

5)
```
int a = 10;
int &p = a;
int &&q = p; //ERRO
```

★ Argument Passing in C++

In C++ Argument passing is of 3 Styles :

1) Pass by value :-

This style of argument passing in c++ is exactly same as of C language.

* In this style, the programmer passes the value of a variable as actual argument which is then copied in the variable declared as formal argument in the function's argument list.

* Now since both the actual and formal arguments have the same value so we can easily access the actual arguments value in formal argument, but since their addresses are different, so any change done

Date
Page

In the formal arguments value is never reflected in the actual argument.

2) Pass by address :—

* This style of argument passing in C++ is same as pass by reference of C Language.

* Here the programmer passes the address of a variable as actual argument, which is then recived by a pointer declard as formal argument in the function's argument list.

* Now since we have pointer to the actual argument, it becomes possible for us not only to access the value of actual argument, but also we can change the value of actual argument.

* But, since we are using pointers, our code will become difficult as with pointers we have to use Indirection Operator.

* And this makes our code a little but typical to read and understand.

* To overcomes the problem of readability, C++ has introduced a new style of passing arguments which is called Pass By Reference using Reference variables.

3) Pass By Reference Using Reference Variables:-

* This style of argument passing has been introduced by C++ only and was not present in C language.

* In this style we pass the value of a variable as actual argument, but received it in a Reference variable declared in the function's argument list as formal argument.

* Now since a reference variable is an Alias to the variable to which it is referring, so we can not only access the value of actual argument but we also can change it's value.

* Moreover, since reference variable do not use Indirection Operator, so it becomes very easy for us to use/ operate them as compared to Pointers.

* Thus in c++ we can say that we have two ways of implementing pass by reference. one using pointer and another, using reference variable.

☆ Example of Argument passing in c++ :

⇒ Pass by value :

```
#include <iostream.h>
#include<conio.h>
void increment(int);
int main()
{
    int a=10;
    cout<<" Before calling increment a is :"<<a<<endl;
    increment(a);
    cout<<" After calling Increment a is : "<<a;
    return 0;
}
voidd increment(int p)
{
    p++;
}
```

output :-

```
Before calling increment a is : 10
After calling increment a is : 10
```

→ Pass by Address :-

```
#include <iostream.h>
#include <conio.h>
void increment(int *);
int main()
{
    int a=10;
    cout<<" Before calling increment a is :"<<a<<endl;
    increment(&a);
    cout<<" After calling increment a is : "<<a;
    return 0;
}
void increment (int *p)
{
    (*p)++;
}
```

Output :-

```
Before calling increment a is : 10
After calling increment a is : 11
```

⇒ Pass By Reference :-

```
#include<iostream.h>
#include <conio.h>
void increment(int &)
{int main()
{
  int a=10;
```

```
    cout<<" Before calling increment a is : "<<a<< endl;
    increment(a);
    cout<<" After calling increment a is : "<<a;
    return 0;
}
void increment (int &p)
{
    p++;
}
```

Output :-

```
Before calling increment a is : 10
After calling increment a is : 11
```

★ Program :- write a program to create a function called swap() which should accept 2 integer as argument and swap their values. Finally display the swapped values.

" Solution "

```
#include<iostream.h>
#include<conio.h>
void swap( int &, int& )
int main()
{
    int a, b;
    cout<<" Enter 2 int : ";
    cin>>a>>b;
    swap(a, b);
```

```
    cout<<"a= "<<a<<",b "="<<b;
    getch();
    return 0;
}
void swap(int &p, int &q)
{
    int temp;
    temp = p;
    p = q;
    q = temp;
}
```

Assignment :-

WAP to accept radius of a circle from the user in the function main(). Now pass it to a function called calculate(). This function should calculate, area and circumference of the circle, but results should be printed in the function main() only. Assume radius to be an integer value.

Copy Constructor :-

In our previous lecture, we stopped our discussion on the copy constructor topic with word the "Reference" After learning about the reference variable now we again starting the topic "Copy Constructor"

Date
Page

★ Again definition of Copy constructor :-

* A Copy constructor is a special constructor of our class.

* A copy constructor is used for copying one object to another.

* It is called whenever we create a new object and want to initialize it with the value of an existing object.

⇒ How do we create a copy constructor :-

To create a copy constructor in our class, we have to declare a constructor which should accept the reference of the object of it's own class of arguments.

Example :-

```
Class Emp {
        int age;
        char name[20];
        float sal;
public :
        Emp();
        Emp(int, char*, float);
        Emp(int);
        Emp(Emp &);  <-- // copy constructor
```

Date
Page

Compiler provides us 2 compiler by default :-

a) Default constructor :-

with blank body. If the programmer has created any constructor then default constructor will not be given.

b) Default copy constructor :-

with code to copy one object to another. If the compiler has not created any copy constructor, then compiler will supply a default copy constructor, but programmer has created his own copy constructor, then compiler will remove its copy constructor.

Q) Now we again implement our "Box" class question with copy constructor :-

```
#include<iostream.h>
#include<conio.h>
class Box
{
    int l;
    int b;
    int h;
```

```
    public :
            Box();
            Box(int);
            Box(int, int, int);
            Box(Box &);
            void Show();
};

Box :: Box()
{
    cout<< "Enter l, b, h : ";
    cin>>l>>b>>h;
}
Box :: Box(int i, int j, int k)
{
    l = i;
    b = j;
    h = k;
}
Box :: Box(int s)
{
    l = b = h = s;
}
Box :: Box(Box &p)
{
    l = p.l;
    b = p.b;
    h = p.h;
}
Void Box :: Show()
{
```

```
    cout<<l<<", "<<b<<", "<<h<<endl;
}
int main()
{
    Box B1;
    Box B2(10);
    Box B3(5,7,9);
    Box B4(B1);
    B1.show();
    B1.show();
    B2.show();
    B3.show();
    B4.show();

    getch();
    return 0;
}
```

→ what is the difference between the following two lines :

1) `Box B4 = B1;`

V/S

2) `Box B4(B1);`

Both are doing Exactly same task. Object B1 is copying in Object B4.

Date
Page

Overall, in c++ we have 7 types of constructor :-

1) Default constructor.
2) Default copy constructor
3) Copy constructor
4) Non - Parameterized constructor.
5) Parameterized constructor

} we have covered These all.

6) Default Parameterized constructor.
7) Dynamic constructor.

} we will cover it later.

What is default function Argument ?

* Default function Argument is a technique using which a programmer can call a single function in multiple different ways.

* In other words, DFA allows us to write a single function definition ~~and~~ & and call the same function with multiple different number of arguments.

* for a programmer DFA is an alternative to function overloading. That just like function overloading. DFA also allows us to call the function in different ways, but in case of function overloading we define multiple functions and then we can call them differently.

* But in case of DFA we define just one function, and that same function can be called in different ways..

* So DFA is much simpler than function overloading.

* Example :-

```
Void printline(char = "*", int = 5);
int main()
{
    printline("A", 10);  ⟶  printline('A', 10);
    printline('#');  ⟶  printline('#', 5);
    printline();  ⟶  printline('*', 5);
    return 0;
}
```

Restriction on DFA

* Although DFA is powerfull technique but it has one major restriction.

* The restriction is that, the default argument declared in function prototype must always be the Trailing Arguments.

* in other words we can say that if an argument in a function has been declared as default argument than all the arguments

after it must also be declared as default argument or it should be the last argument in function declaration.

* Example :-

i) void show(int, int, int = 30); // OK

ii) void show (int = 10; int, int = 30); // ERROR

iii) void show(int, int = 20, int = 30); // OK

iv) void show(int = 10; int = 20, int = 30); // OK

v) void show (int, int = 20, int); // ERROR.

* Restriction on DFA Call

* while calling a function which has default arguments set, If we skip a particular argument, than all the arguments after it must also be skipped.

Example :-

* Void display(int = 10, int = 20, int = 20);

Date
Page

* which of the following calls are valid ?

display(); // ok → Display(10,20,30);

display(15); // ok → Display(15,20,30);

display(,25,35); // ERROR

display(10,25,35); // ok.

* Default Parameterized Constructor.

* we can abaid constructor overloading using Default Parameterized constructor.

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

class student
{
    int roll;
    char grade;
    float per;
  public:
    student(int=0, char=' ', float=0.0);
    void get();
    void show();
};
```

```
Student :: student ( int r , char g , float p )
{
    roll = r;
    grade = g;
    per = p;
}
void student :: get()
{
    cout<<" Enter roll, grade and per : ";
    cin>>roll>>grade>>per;
}
void student :: show()
{
    cout<<roll<<","<<grade<<","<<per<<endl;
}
void int main()
{
    student s(10, 'A', 87.3);
    student p;
    p.get();
    s.show();
    p.show();
    getch();

    return 0;
}
```

"output"

```
Enter roll, grade and per : 18 B 56
10 A 87.3
18 B 56
```

Date
Page

★ What is a Destructor ?

* A Destructor is special member function of a class having the same name as the class but prefixed with a symbol of tilde (~).

* for example if the name of the class is Emp, then its destructor will be :-
~Emp();

Q→ What is the destructor called ?

* whenever the c++ compiler decides to destroy an object, then just before removing that object from memory, the compiler automatically calls the destructor function abvaiable in the class.

* That means in c++, every object in its entire lifetime at the minimum calls 2 member functions.

* These are :

1. Constructor : called immediately after the object gets created.

Date
Page

2. Destructor : called just before the object is to be destroyed.

Can we say that a destructor destroys objects ?

Ans :- We would never say this !!

* A destructor never destroys an object. Destruction of an object is carried out by the compiler or the OS.

* A destructor is just a symbolic representation that the object is about to be destroyed.

A program to Demonstrate working of A Destructor :-

```
#include<iostream.h>
#include<conio.h>
class Emp
{
    int age;
    char name[20];
    float sal;
public :
        Emp();
        void show();
        ~Emp();
```

```
};
Emp :: Emp()
{
    cout<<" Enter age, name and sal : ";
    cin>> age >>name >> sal ;
}
void Emp :: show()
{
    cout<< age <<" "<< name<<" , "<< sal <<endl;
}
Emp :: ~Emp()
{
    cout<< name <<" ; ";
    cout<<" object destroyed ! "<<endl;
}
int main()
{
    Emp E, F;
    E. show();
    F. show();
    getch();
    return 0;
}
```

Output :—

```
Enter age, name and sal : 24 Ravi 32000
Enter age, name and sal : 21 Nitin 30000

24 Ravi 32000
21 Nitin 30000
```

Date
Page

Nitin : object destroyed
Ravi : object destroyed.

★ Comparison b/w Constructor and Destructor

Constructor	Destructor :
1) They are special member function of the class having the same name as that of the class.	It is also a special member function of a class, having the same name as that of the class but prefixed with the symol of the tilde (~).
2) They are called automatically as soon as the object of a class gets created i.e. their calling is implicitly done by the compiler.	They also are automatically called as soon as the object is to be destroyed i.e. their calling also is implicitly done.
3) They are called in the same order as objects are created.	Destructor is called in the reverse order of creation of the object.
4) Constructors can be parameterized.	A destructor can never be parameterized.
5) Since they can be parameterized, so we can overload them and thus a class can have multiple constructors	As it doesn't accept so we can't overload it and thus a class can just on destructor.

6)	If we do not define any constructor ourself, then we will get 2 constructors from compiler, called as default constructor and default copy constructor	if we do not define any destructor in our class then the c++ compiler automatically provide only one destructor called.
7)	The default constructor has a blank body while the default copy constructor has statements for copying one object to another in its body.	The default destructor also has a blank body.
8)	Constructors can not be made static.	A destructor also can not be made static.
9)	Constructors can not be declared as "const".	A destructor also can't be declared as "const".
10)	Constructors are not inherited.	A destructor also is not inherited.
11)	A constructor can't be declared as "virtual".	A destructor can be declared as "virtual".

Q) What is getline() function ?

* The cin is an object which is used to take input from the user but does not allow to take the input in multiple lines.

* To accept the multiple lines, we use the getline() function.

* It is a pre-defined function defined in a <string.h> header file used to accept a line or a string from the input stream untill the delimiting character is encountered.

⇒ Syntax of calling getline():

cin.getline(<char_arr_name>, <max_no_char>);

Ex:-

```
char str[10];
cin.getline(str, 10);
```

It will either stop at Enter or At 9th character whichever occurs first

A program to Demonstrate Benefits of Destructor

```
#include<iostream.h>
#include<string.h>
#include<alloc.h>
Class Emp
{
    int age;
    char *p;
    float sal;
public:
    Emp();
    void show();
    ~Emp();
};
Emp: Emp()
{
    cout<<" Enter age and sal:";
    cin>>age>>sal;
    char name[20];
    cout<<" Enter your name : ";
    cin.ignore();
    cin.getline(name, 20);
    int x = strlen(name);
    p=(char*)malloc((x+1)* sizeof(char));
    strcpy(p, name);
}
void Emp :: show()
{
  cout<<age<<","<<p<<","<<sal<<endl;
}
```

```
Emp :: ~Emp()
{
    free(p);
}
int main()
{
    Emp E;
    E. show();
    return 0;
}
```

Output :-

```
Enter age and Sal : 25 32000
Enter your name : Amit

    25, Amit, 32000.
```

☆ what is Modern C ++ ?

* C++ is one of the most widely used programing language in the world.

* It's been around from the last 40 years and was initially much like C language due to backward compatibility

* But to meet modern computing requirements it was redesigned in the year 2011 and the name ~~Moodern~~ Modern C++ was given.

Date
Page

* After C++ 11, two never vaxsions of the standards have been accepted.

C++ 14: which is a bugfix to C++ 11.

C++ 17: which adds some more new features to the languages.

* So now the term Modern C++ refers to C++ 11, C++ 14, and C++ 17.

Little History of c++ 11/14/17 ?

* C++ 11 is a version of the standard for the programming language C++.

* It was approved by the International organization for standardization (ISO) on 12th ~~Agus~~ August 2011, replacing classic C++, and superseded by C++ 14 on 18th August 2014.

* Then on 1st Dec 2017, C++ 17 was released with some new modules and features.

which compiler support C++ 17 ?

Date
Page

* All modern compilers except Turbo C++ 3.0 are now fully supporting C++ 17 standards.

* The most popular compilers are GCC C++, Clang C++, and Microsoft C++ and following table lists their support for C++ 17.

Compiler	C++ 17 Support:
⇒ GCC 8	Full Support for C++ 17
⇒ Clang C++	Implements all the features of C++ 17
⇒ Microsoft C++	supports almost all of C++ 17.

which compiler we will use ?

* Since we are using Code Blocks IDE which has support for GCC compiler so we will be using GCC.

* Moreover, since there is not much difference in C++ 14 and C++ 17 so any version of code blocks which supports them will work.

* C++ 14/17 has lot of new concepts which were not present in classic C++.

* All these new features make Modern C++ programs type-safe and easier to write, extend and maintain.

* These are listed on the next slide

☆ Modern C++ features :-

"NEW TOPICS"

* New concepts of header files	* Lambda Expressions
* Namespaces	* Rvalue References
* Automatic type declaration	* Smart pointer, Unique pointer, Shared pointer.
* New syntaxes of initialization	* Static cast, Reinterpret cast, Dynamic cast.
* Deleted functions	* Exception Handling.
* Nullptr	* Standard Template Library.
* Delegating constructor	* override and final
* The string class	

☆ What is Code Block ?

* Code :: Blocks is a free, open-source cross-platform IDE (not a compiler) which was designed to support C and C++ language.

* However now it also supports other languages including Fortan and D.

Date
Page

what compiler Code Blocks uses ?

* Code :: Blocks supports multiple compilers like :

* Gcc , MinGW , Digital Mars , Microsoft Visual C++ , Borland C++ , LLVM Clang , watcom , LCC and the Intel C++ compiler .

Important Difference B/w standard C++ and C++ 11 / 14 / 17 .

1) The names of all C++ header files do not contain .h extension.

2) So <iostream.h> becomes <iostream> , <fstream.h> becomes <fstream> and so on ..

3) for C language header files we still have the .h convention available but recommendations are to use these header file names prefixed with the letter 'c' and dropping '.h'.

4) So <math.h> becomes <cmath> , <stdlib.h> becomes <cstdlib> and so on ..

5) All predefined object like cout, cin etc are now placed inside something called namespace.

6) The default namespace provided by C++ is called std.

7) So instead of writing cout, the statement now becomes std :: cout.
Ex :-

```
std :: cout<<" Hello world ";
```

What is Namespace ?

Before we discuss exactly what a namespace is, it is probably best to consider a simple example of when and why we would need them.

Consider the following code :

```
# include<iostream>
int main() {
    int value;
    value = 0;
    double value; <- ERROR HERE!
    value = 0.0;
    std :: cout<< value;
    return 0;
}
```

Date
Page

* In each scope, a name can only represent one entity. So, there cannot be two variables with the same name in the same scope.

* Using namespace, we can create two variables or functions having the same name.

Syntax :-

```
namespace < some name>
{
    edtities
}
namespace first
{
    int val;
}
namespace second.
{
    double val;
}
```

```
int main()
{
first::val=10;
second::val=15;
return 0;
}
```

Points To Remember. :-

1) Namespace is a logical compartment used to abaid naming collisions.

2) Default namespace is global namespace and can access global data and functions by proceeding (::) operator.

Date
Page

24.8 20.07.

24.2 ... 01

3) We can create our own namespace and anything declared within namespace has scope limited to namespace.

4) Creation of namespace is similar to creation of class.

5) Namespace declarations can appear only at global scope.

6) Namespace declarations don't have access specifiers. (public or private)

7) No need to give semicolon after the closing brace of definition of namespace.

☆ The Namespace "STD" :-

* The built in c++ library routines are kept in the standard namespace called std.

* That includes stuff like cout, cin, string, vector etc.

* Since they are in namespace std so we need to apply the prefix std:: with each one of them.

* Thus cout becomes std::cout, cin becomes std::cin and so on

```
#include <iostream>
int main()
{
int a,b,c;
std::cout<<"Enter 2 int "<<std::endl;
std::cin>>a>>b;
c = a+b;
std::cout<<"Nos are "<<a<<" and "<<b<<std::endl;
std::cout<<"Their sum is "<<c;
return 0;
}
```

Avoiding "std"

* The keyword using is used to introduce a name from a namespace into the current declarative regions.

* So using namespace std means that we are going to use classes or functions (if any) from "std" namespace.

* Thus we don't have to explicitly call the namespace to access them.

Modified version :-

```
#include<iostream>
using namespace std;
int main()
{
    int a, b, c;
    cout<<"Enter 2 int"<<endl;
    cin>>a>>b;
    c = a+b;
    cout<<Enter 2 Nos are "<<a<<" and "<<b<<endl;
    cout<<"Their sum is "<<c;
    return 0;
}
```

★ New Data Types Added

* int : upgraded to 4 bytes.
* bool : a special type of 1 byte in size for storing true/ false.
* wchar_t : a special data type for storing 2 bytes of characters.
* long long : a new data type of integer family supporting size of 8 bytes.

★ The "Auto" Keyword :-

* In Modern C++, the compiler can automatically determine the data type of a variable at the point of declaration using it's initialization expression :

* So,
```
int x = 4;
```

* can now be replaced with

```
auto x = 4;
```

* This is called automatic type deduction..

* Also we can write
```
Char *p = "Bhopal";
```
as
```
auto p = "Bhopal";
```

* This also applies to pointers of class type.

* So,
```
Box *p = new Box;
```
can also be written as
```
auto p = new Box;
```

* The auto keyword can also automatically detect a function's return type.

* For example :

```
auto add (int x, int y)
{
    return x + y;
}
```

* since x + y evaluates to an integer, the compiler will detect this function should have a return type of int.

* Modern C++, supports three basic ways to initialize a variable :

* Copy initialization
* Direct ~~init~~ initialization
* Uniform ~~int~~ initialization.

* Copy initialization :- is done using assignment operator

```
int n = 5; // copy intialization
```

* Direct initialization :- is done by using paranthesis.

```
int n(5); // direct intialization
```

* Uniform initialization :- is done using curly braces

```
int n{5}; // uniform initialization
```

If we leave the braces blank then compiler initializes the variable to 0.

For Examples :-

```
int n{};
cout<<n; // will display 0
```

* Modern C++ allows us to initialize data members at the point of declaration and this is called In-class initialization

```
Class Circle
{
  int radius = 10;   // OR, from C++ 14 onwards
};
```

But, remember if we do not initialize them, their default value will still be garbage.

☆ Range Based for Loop.

* Modern C++ provides us an easy way to traverse an array called "Range Based for Loop".

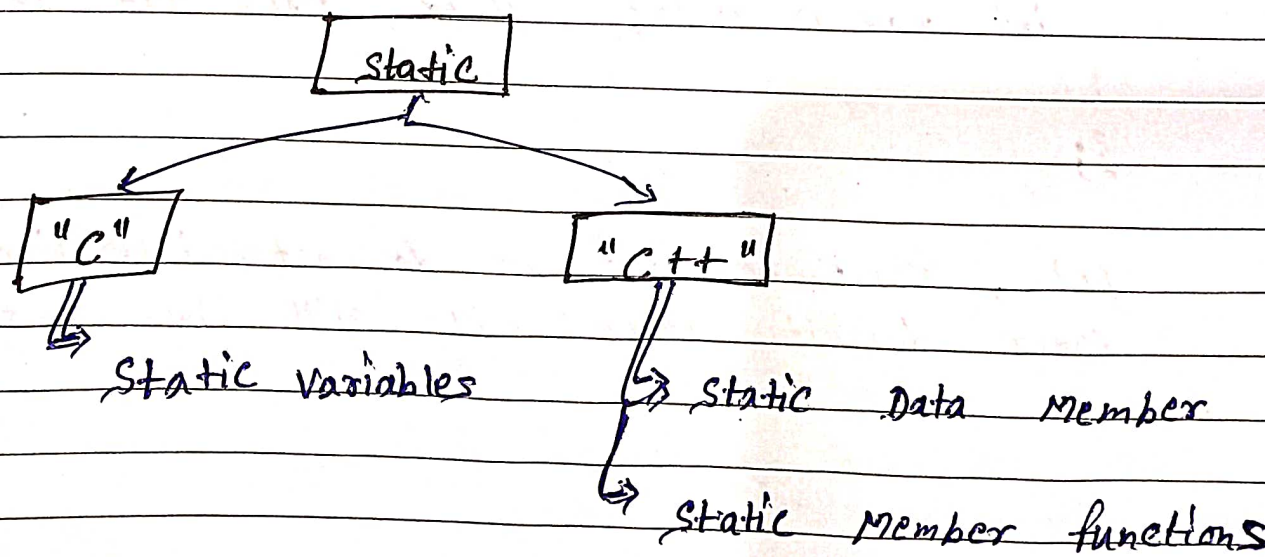
* Syntax :

```
for (<data type> <var_name> : <array_name>)
{
  // Loop body
}
```

* Example :

```
int arr[] = {10, 20, 30, 40};
for (int x : arr)
    cout<< x <<endl;
```

☆ Using The Keyword Static



★ What is static in C Language ?

* "static" is called as storage class in C and does two things :

1) The variable automatically gets initialized with 0 and not garbage. If the variable is of type float the it will be initialized with 0.0. and if it is of char type then it will be initialized 0 of ASCII which is ASSCII of "/0".

2) Once declared, they stick in memory until the program finishes. That is if a static variable inside a function body the even if the function's execution gets over, still the static variable will survive in RAM and will only get destroyed when the program finishes.

Guess the output :—

```
void show();
void main()
{
    show();
    show();
    show();
}
void show()
{
    static int a;
    printf("a = %d ", a);
    a++;
}
```

→ This line Executes only once

Output :— 0
1
2

* when a static is used, variable or data members or Member functions can not be modified again.

* It is allocated for the lifetime of program.

* Static variables are initialized only once.

* Computer persist the variable till the end of the program.

* Comparision between NON-Static and Static :-

	Non-static	Static
1)	They are called as instance variables.	They are called as shared variables or class variables.
2)	They have a separate copy created for every object of the class.	They have a single copy created amongst all the objects.
3)	They are located/unloaded in RAM along with objects.	They arrive in memory even before any object gets created and they stay in memory through out the execution of the code. They are unloaded from memory only when the program gets over.

4) No separate declaration/ Re-declaration is required for them as they are brought in memory along with object.	They must be declared outside the class using the following syntax : <data type><class_name>::<static_var_nam this is because they require memory even before any object gets created and this statement is required for it.
5) They can only be accessed using the object of the class since they are a part of the object	They too can be accessed using object but the correct way is to access them using class name. Moreover if we try to access them using object then the compiler does not generate any error, but changes our statement to the correct form that using the chass name.

Assignment :- WAP to create a class Emp with the following data members.

i) age : This should be of type int and should store age of the Emp.

ii) Name : This should be char[] and should store name of the Emp.

3) count :- This should be of type int and should keep a track of number of employees working in the company.

Also provide following member functions in your class

1) A parameterized constructor for initiating age and name.

2) A member function called show() to display age and name.

3) A member function called showcount() which will display number of employees currently working in the company.

Solution:- A program To Demonstrate Benefits of Destructor.

1) version 1 :-

```
#include <iostream.h>
#include<cstring.>
using namespace std;
class Emp {
        int age;
        char name[20];
        static int count;
public :
        Emp( int, char* );
        void show();
        void showcount(); };
```

```
int Emp :: count ;
Emp :: Emp (a char , char *p )
{
        age = a;
        strcpy (name, p);
        count ++;
}
void Emp :: show()
{
   cout<<age " , " << name <<endl;
}
void Emp :: showcount(); {
cout<<" Total ~~more~~ emp working are " << count<<endl;
 }
int main()
{    Emp E( 1 ,"Amit");
     Emp E(24, "Sumit");
     Emp G( 22, "Ravi");

     E. show();
     F. show();
     G. show();

     E. showcount();
     F. showcount();
     E. showcount ();

     return 0;
}
```

Output :-

```
21 Amit
24 Sumit
22 Ravi
Total emp working are 3
Total emp working are 3
Total emp working are 3.
```

2) version 2 :-

```
#include<iostream.h>
#include<cstring>
using namespace std;
class Emp {
        int age;
        char name[20];
        static int count;
  public:
        Emp(int, char*);
        void show();
        void showcount();
        ~Emp();
};
int Emp::count ;
Emp ::Emp(int a, char *p)
{
   age =a;
   strcpy(name, p);
   count++;
}
void Emp ::Show()
{
   cout<<age<<", "<<name<<endl;
}
Void Emp :: Show count
{
   cout<<" Total emp working are "<<count<<endl;
}
```

```
Emp :: ~Emp() {
        count--;
}
int main() {
        Emp E(21, "Amit");
        Emp F(24, "Sumit");
        Emp G(22, "Ravi");
         E.show();
        F.show();
        G.show();
        E.showcount(); {
        Emp x(21; "Deepak");
        Emp y(1,"Jyoti");
        x.show();
        y.show();
        x.showcount();
         }
       E.showcount();
       return 0;
}
```

Output :—

```
21 Amit
24 Sumit
22 Ravi
Total emp working are 3
21 Deepak
21 Jyoti
Total emp working are 5
Total emp working are 3
```

"Static Member Functions"

1) Static member function is a special member function of a class which is declared using the keyword "static".

2) Whenever we define a static member function we never use the keyword "static".

3) To call "static" member function we require the name of the class and not the name of the object.

4) Syntax of calling these functions is :

<class_name> :: <member_fun^n_name>();

5) We should make those member functions as static within which we are only accessing "static" data and not any non-static data.

6) If we declare a member function as static the we cannot access any non-static data within the member function.

7) We can declare every member function of the class as static but we can never declare a constructor or destructor of a class as static. This is because constructors and destructors are closely bound to object within a static member function has no connection with the object.

Assignments :- Make the following changes in your code :

1) Add a member called "salary" of type float.

2) Define a member function called showavgsal() which whenever called displays the average salary of the company.

* Using the keyword "inline"

```
int square(int n)
{
    return n*n;
}
int main()
{
  int a=10, b=20;
  int c;
  c= square(a);
  cout<<" square of "<<a<<" is "<<c<<endl;
  c= square(b);
  cout<<" square of "<<b<<" is <<b<<endl;
   return 0;
}
```

Overheads, the compiler has to face when it solves a function call :

1) Prototype checking

2) Argument passing

3) Pushing the address in Stack.

4) Executing the function.

5) while returning back it has to pop the top address from the stack and then it resume the remaining part.

Q→ what are the inline Functions ?

* An inline function is a function which is prefixed with the keyword inline during its definition.

* The general syntax of declaring an inline function is :

```
inline <return_type><fun_name>(<list_of_arg>)
{
    // body
}
```

for Example :-

```
inline int add(int a, int b)
{
    return a+b;
}
```

* when we make function as inline then the compiler copies its definition in place of its call. In other inline functions are those whose call gets replaced by its definition in the machine code.

* Due to this, number of overhead of the compiler reduces since it does not has to perform any activity related to a function call like, prototype checking, argument passing and most importantly stack maintenance.

Example:-

```
inline int Square(int n)
{
    return n*n;
}
int main()
{
    int a=10; int b=20;
    int c=;
    c = Square(a);
    cout<<" Square of "<<a<<" is "<<c<<endl;
    c = Square(b);
    cout<<" Square of "<<b<<" is "<<c<<endl;
    return 0;
}
```

* There are 3 important points we must remember, before declaring a function as inline :.

1) If a function is declared as inline then it must have short and small definition, i.e. number of lines in an inline function must be very less.

2) The definition of an inline function must be very simple and should not contain any complex logic like loop, break, continue etc.

3) The definition of an inline function, must appear in the code before the function call. In other words the compiler must be aware about the functions inline nature even before it encounters function call.

If any of the above points or rules are violated then the compiler simply ignore. The keyword inline and handles the function in a normal manner.

Types :- Types of inline functions :-

1) Explicit inline. :- These are those functions in which the programmer himself mentions the keyword inline while defining the functions.

2) Implicit Inline :- These are those functions which are Automatically declared as inline by the compiler.

Implicit inline :- Implicit inline functions are those functions, whose definition is written within the body of the class and even if we do not use the keyword "inline" with them, still the C++ compiler assumes them to be "inline." function

All compiler generated functions are always implicit inline functions.

Explicit inline :- Explicit inline functions are those, which are declared within the class, defined outside the class but while defining them the programmer prefixes them with the keyword "inline".

Such functions are not automatically treated as inline, rather on programmer's request the compiler takes them to be inline.

Note :- All the rules for inline functions, which we have discussed before must be allowed here also. Also if these rules are broken the here too the compiler will simply

ignore the keyword inline and handle the function in a normal way.

Thus we can say that making a functions inline is simply a request made by the programmer to the compiler which can either be accepted or rejected, based upon the circumstances by the compiler.

Program :-

```
class student
{
    int roll;
    char grade;
    float per;
public :
    void get()
    {
    cout<<"Enter roll, grade and per:";
    cin>>roll>>grade>>per;
    }
    void show();
};
inline void Emp :: show()
{
    cout<<roll<<", "<<grade<<", "<<per<<endl;
}
```

implicit inline

Explicit inline.

Note :- All the functions (default constructor, default copy constructor and default Destructor) which are generated by the compiler are also implicit inline functions.

★ The "this" Pointer.

```
#include<iostream>
using namespace std;
class Student {
        int roll;
        char grade;
        float per;
public:
        void get();
        void show();
};
void Student :: get()
{
  cout<<"Inside get"<<endl;
  cout<<"Address of my calling object is "<<this;
}
void Student :: show()
{
    cout<<"inside show"<<endl;
    cout<<"Address of my calling object is "<<this;
}
int main()
{
 student s;
 cout<<"In main address of s is "<<&s<<endl;
 s.get();
s.show();
student p;
cout<<"in main address of p is "<<&p<<endl;
p.get();
```

P. show();
return 0;
}

Output :-

In main address of s is 0x61fc14
inside get
Address of my calling object is 0x60fe14
inside show
Address of my calling object is 0x61fe14
in main address of p is 0x61fe08
inside get
Address of my calling object is 0x61fe08
inside show
Address of my calling object is 0x61fe08

Q) what is "this"?

* In C++, whenever we call a non-static member function using an object, then automatically the C++ compiler passes the Address of the calling object to this member function as argument.

* Inside the member function, this address is stored inside a special pointer called as the "this" pointer.

* So, we ~~say~~ can say that every non-static member function in C++ language, always knows the caller address via its "this" pointer.

Q) what is the data type and size of "this" ?

* Since the type of pointer is always same as the type of the variable to which is pointing. and we know that "this" pointer always points to the calling object.

* So the data type of "this" will also be same as the data type of object, which is nothing but the class name. In our example the type of "this" will be student.

* The size of "this" is same as the size of any pointer on the underlying platform. In our case since we are using GCC compiler so "this" will be 4 bytes and if we use TC the "this" will be of 2 bytes.

☆ Accessing Object member using "this" :-

```
#include <iostream>
using namespace std;
class student {
        int roll;
        char grade;
        float per;
public :
        void get();
        void show();
};
```

```
void student :: get() {
    cout<< Enter roll, grade and per : " ;
    cin>> this→roll >> this→grade >> this→per ;
}
void student :: show() {
    cout<< this→roll<<", "<< this→grade<<", "<< this→per ;
}
int main()
{
    student s, p ;
    s.get() ;
    p.get() ;
    s.show() ;
    p.show() ;
    return 0 ;
}
```

★ Benefits of using "this" :—

1) Using "this" we can resolve the overlapping of class members done by local variables of the same name inside a member function.

2) Using "this" we can reduce number of line in copy constructor to just 1 single line.

3) Using "this" we can reduce the number of statements inside the body of overloaded operator function.

Benefit → 1

```
#include <iostream>
# include <cstring>
Using namespace std;
class Emp {
        int age;
        char name[20];
        float sal;
public :
        Emp( int, char* , float );
        void show();
};
Emp :: Emp( int age , char * name , float sal )
{
    this → age = age ;
    strcpy ( this →name , name ) ;
    this →sal = sal ;
}
void Emp :: show() {
        cout<< " age= " <<age<< " name= " <<name<< " sal = "
            <<sal ;
}
int main() {
        Emp e(21 , " Amit " , 30000.0)
        return 0 ;
}
```

Benefit → 2

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

```
Class Box
{
    int l, b, h, w;
Public :
        Box( int, int, int, int);
        Box( Box&);
        void show();
};
Box :: Box ( int l, int b, int h, int w) {
      this -> l = l;
      this -> b = b;
      this -> h = h;
      this -> w = w;
}
inline Box :: Box (Box &p) {
      *this = p;
}
void ~~show~~ Box :: show() {
   cout<< "len = "<<l<<", breadth = "<<b<< ", height = "<<h<<
        ", weight = "<< w << endl;
}
int main() {
      Box B1( 10, 20, 30, 40);
      Box B2(B1);
      B1. show();
      B2. show();
      return 0;
}
```

★ Using The Keyword "Const" :-

* consider the following Example :-

```
int main()
{
    int red = 10;
    float pi = 3.14;
    double area, circ;
    area = ++pi * red * red;
    circ = 2* ++pi * red;

cout<< "Area = " << area << endl;
cout<< "circumf = " <circ << endl;
return 0;
}
```

This statement is logically wrong, but still the code will compile and run, so if we don't want compiler to ignore this, then we must use the keyword, "const".

Q→ what is "const" ?.

* Const in C++ is a Keyword.

* whenever we don't want to change a variables value even accidently, then to be double sure about it, we must declare the variables as "const" variable.

```
int main()
{
    int rad = 10;
    const float pi = 3.14;
    double area, circ;
    area = ++pi * rad * rad;
    circ = 2 * ++pi * rad;
```

Syntax Error.

```
cout<<"Area=" <<area<<endl;
cout<<"circumf=" <<circ;
return 0;
}
```

* Points to remember with "const":

1) A "const" can never be changed, i.e. throughout its lifetime the value will remain same and any attempt to change the value will give Syntax Error.

2) when we declare a variable of "const", then we must initialize it, at the point of declaration.

Places where "const" can be used:-

1) "const" variable.
2) Pointer to "const".
3) "const" pointer.
4) "const" pointer to "const".
5) "const" function arguments.

6) "const" data members.
7) "const" member functions.
8) "const" objects.

Pointer To "const"

* we can declare pointer to const in two ways :

1. const <data_type> * <ptr_name> ;

2. <data_type> const * <ptr_name> ;

For Example :-

```
const int *p ;
int const *p ;
```

Points to remember :

1) A Pointer to const can never change the value of the variable to watch it is pointing.

2) However it can be reinitialized to print to some other address at any time in the code.

```
int main()
{
    int a=10, b=50;
    const int *p;

p = &a;
cout<<a<<", "<<*p<<", "<<b<<endl;

a = 20;
cout<<a<<", "<<*p<<", "<<b<<endl;

*p = 30;
cout<<a<<", "<<*p<<", "<<b<<endl;

p = &b;
cout<<a<<", "<<*p<<", "<<b<<endl;

b = 60;
cout<<a<<", "<<*p<<", "<<b<<endl;

*p = 70;
cout<<a<<", "<<*p<<", "<<b<<endl;

return 0;
}
```

Const Pointer :-

1) A const pointer can change the value of the variable to which it is pointing.

2) However it can **not** be reinitialized to some other address.

3) This means that, throughout its lifetime it will stick with the same variable.

4) Till now we have seen const pointer in action with "this", "array" and "reference variable" All three of them are by default considered as "const" pointers by the compiler.

```
int main () {
        int a=10, b=50;
        int * const p = &a;

cout<<"a<<", "<<*p<<", "<<b<<endl;
a = 20; ✓
cout<<a<<", "<<*p<<", "<<b<<endl;
*p = 30; ✓
cout<<a<<", "<<*p<<", "<<b<<endl;
p = &b; ✗
cout<<a<<", "<<*p<<", "<<b<<endl; ✗
p=60; ✓
cout<<a<<", "<<*p<<", "<<b<<endl;
*p = 70; ✓
cout<<a<<", "<<*p<<", "<<b<<endl;
 return 0;
}
```