Unit - I

Object oriented Modeling and Design

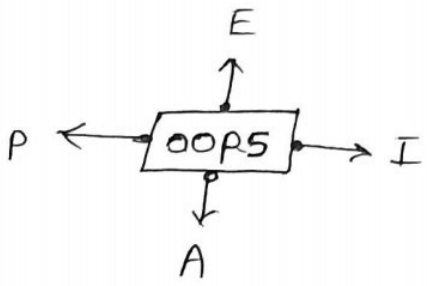
Object oriented Programming (OOPS) concept

Object oriented Programming is a technology or methodology in which all the programming is done with the help of object and class. OOPS allow decomposition of program into a number of entites called objects. object has two parts data in the form of attributes and code in the form of methods or functions. A class is a templet/ blueprint for creating an objects.

The advantages of object oriented programming over procedural programming technique is that they enables programmer to create that do not need to be changed when a new type of object is added.

There are some of the following concept that are used in object oriented programming.

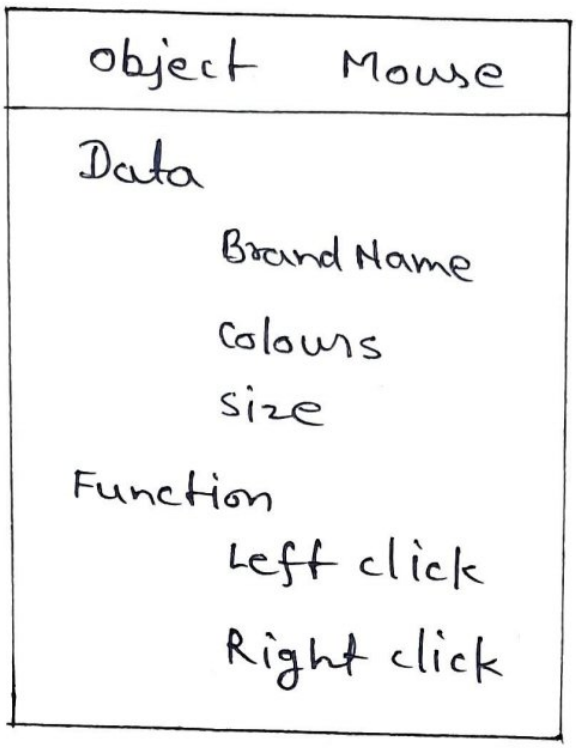
1. objects
2. class
3. Data Abstraction
4. Encapsulation
5. Inheritance.
6. Polymorphism.



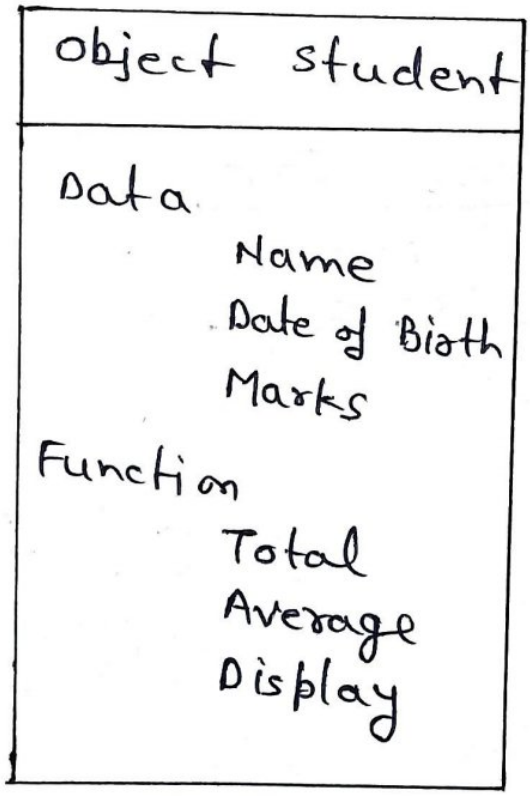
Objects –

objects are the real world entity in an object oriented system. They may represent a person, pen, chair, table, student etc. An entity that has state and behaviour is known as an object. In other words we can say that objects are the variables of class.

For example – computer mouse is an object. It is consedered an object with state and behaviour. Its state would be its colour, size and brandname and its behaviour would be left click, right click.



other Example —



class —

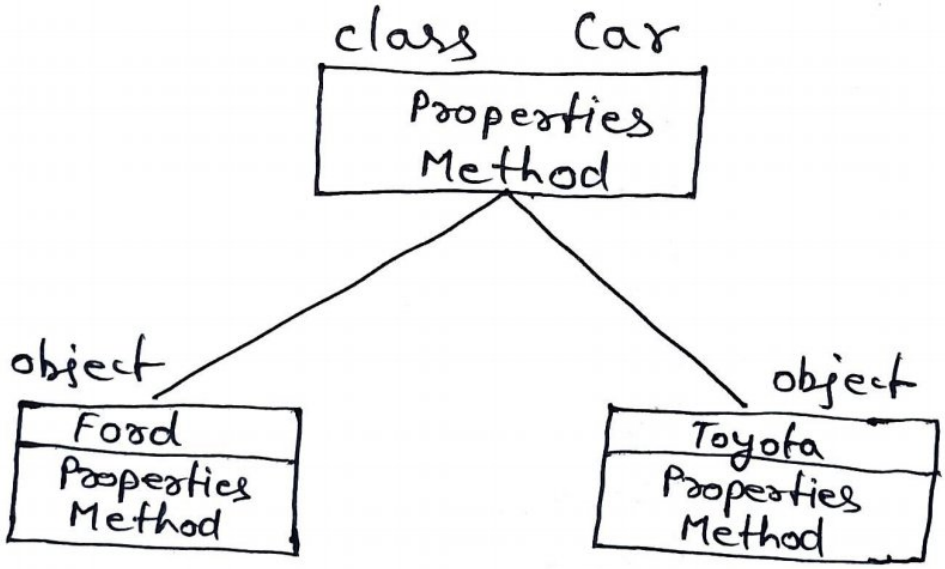
A class is a templet / blueprint for creating an objects. A class is a collection of data member and member functions. Once a class has been defined we can create any number of objects belonging to that class.

For example — Mango, Apple and orange are member of class Fruit.

Ex

```
Fruit Mango;
```

other example.



Syntax of class in c++

```
class classname
{
  data member;
  member function;
};
```

Data Abstraction →

Abstraction refers to the act as representing esential features without including the background details. i.e to represent the needed information in program without presenting the details. for ex- a mobile, which you can turn on- off, adjust volume, play vedio, audio and calling. But you do not know its internal details that is how its receives singnal over the air. How IC is working - since the classes use the concept of data Abstraction, they are known as Abstract Data types (ADT)

Encapsulation —

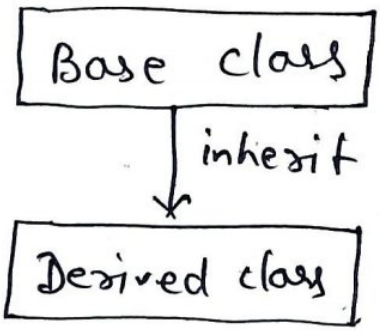
The wrapping up of data and functions into a single unit (called class) is known as Encapsulation. Data Encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those function which are wrapped in the class can access it.

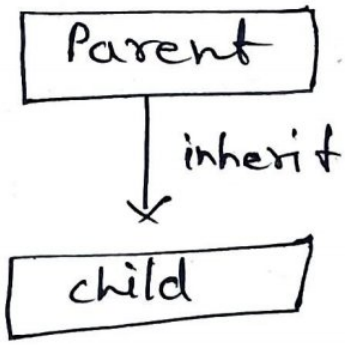
These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding.

Inheritance —

Inheritance is a process or mechanism by which we can access the features (properties) of base class (super class) into its derived class (sub class).
In other words we can say that the reusability of code is know an Inheritance. This means that we can add additional features to an existing class without modifying it.

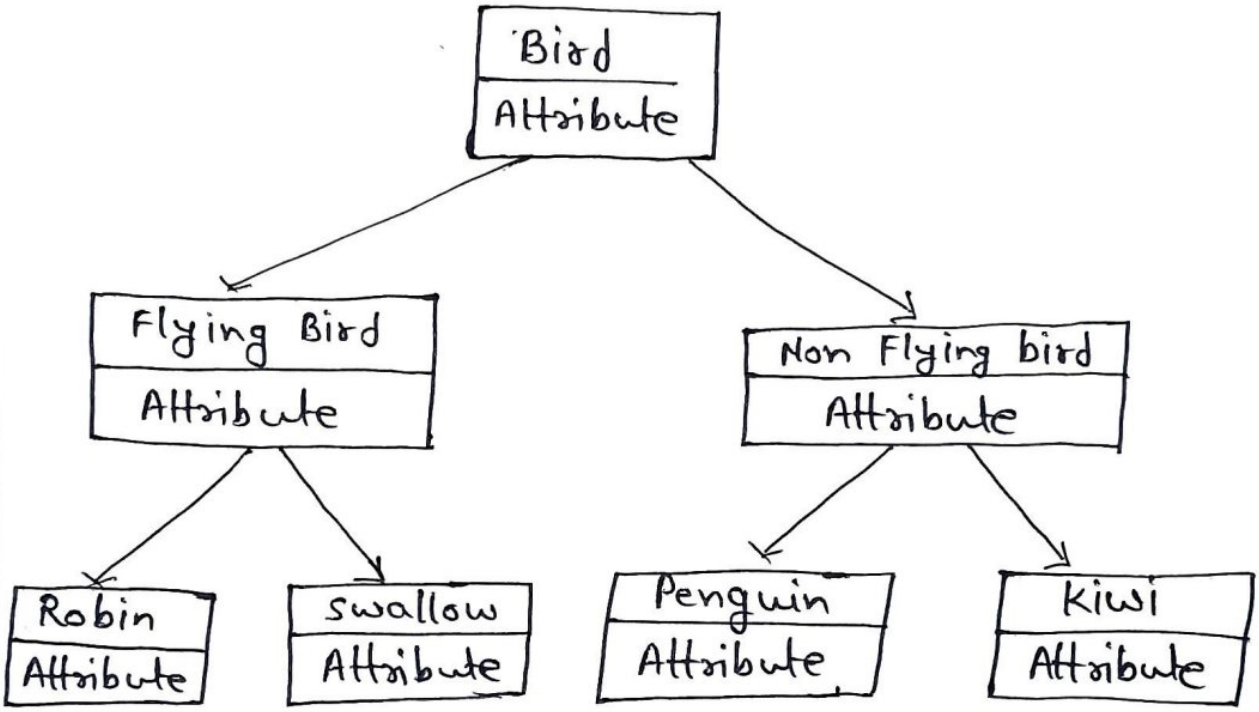
For Ex -





Example -

The bird 'robin' is a part of the class 'flying bird' which is again a part of class 'bird'.



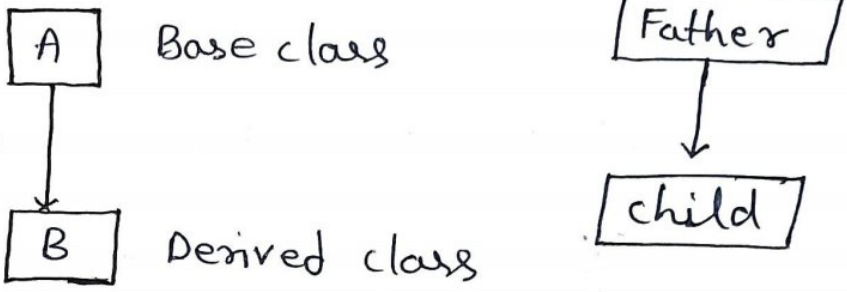
Types of Inheritance

There are some of the following types of Inheritance.

1. Single Inheritance.
2. Multilevel Inheritance.
3. Hierarchical Inheritance.
4. Hybrid Inheritance.
5. Multiple Inheritance.

1. Single Inheritance — When a derived class inherits the properties of single base class is known as single Inheritance.

Example



Here B is a Derived class which is derived from base class A.

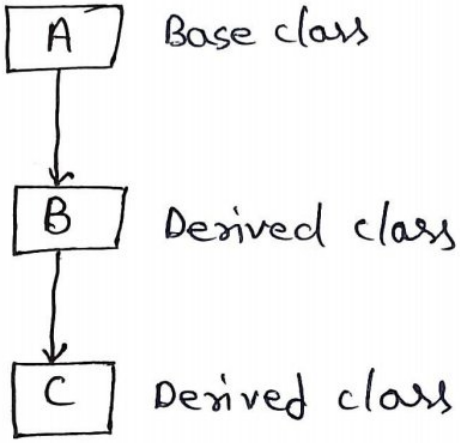
Syntax in c++

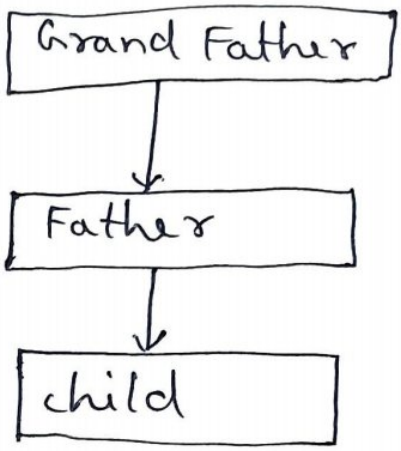
```
class  A <Base class>
{
   body of class A
};

class B <Derived class> : <access-specifier> A
                                    <base class>
{
   body of Derived class B
};
```

2 Multilevel Inheritance — In multileve Inheritance, a derived class is created from another derived class.

Ex





Here C is a Derived class which is derived from Derived class B.

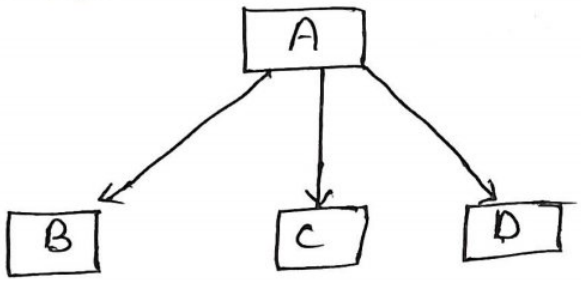
Syntax

```
class A
{
   Body of class A
};
class B : public A
{
   Body of Derived class B
};
class C : public B
{
   Body of Derived class C
};
```

3. Hierarchical Inheritance —

When more than one derived class inherit the features of single base class is known as hierarchical Inheritance.

Example



Base class

Derived class

Here B, C, D are derived class, which are derived from single base class A.

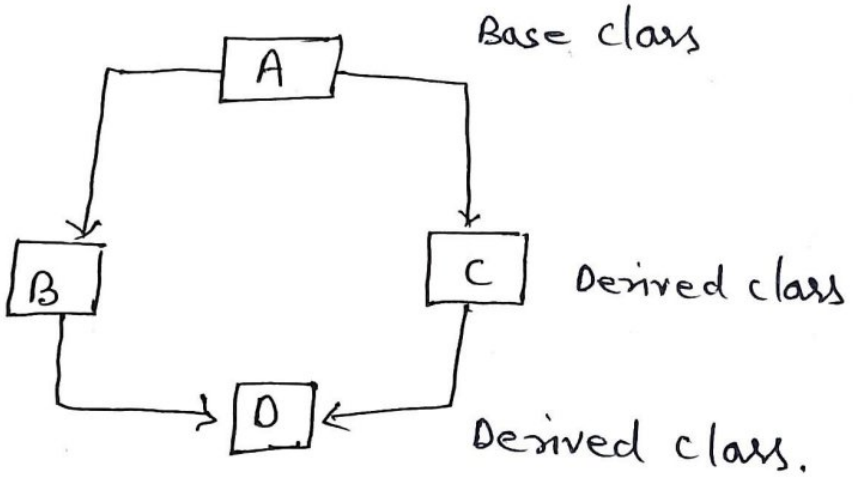
syntax

```
class A
{
   Body of class A
};
class B : public A
{
   Body of Derived class B
};
class C : public A
{
   Body of Derived class C
};
class D : public A
{
   Body of Derived class D
};
```

4. Hybrid Inheritance — Hybrid Inheritance is implemented by combining more than one type of inheritance.

For example combining Hierarchical Inheritance and Multiple Inheritance.

Example



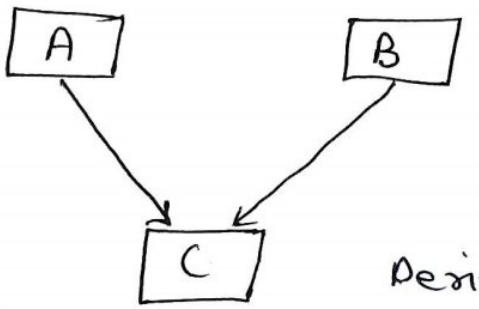
Here D is a Derived class which is derived from two derived class B and C.

syntax

```
class A
{
- - - -
};
class B : public A
{
- - - -
};
class C : pubic A
{ - - - -
};
class D : public B, Public C
{ - - - - -
};
```

5. Multiple Inheritance — When a derived class is inherited from more than one Base class is known as Multiple Inheritance.

Ex



Base class

Derived class

- Here C is a Derived class which is derived from two base class A and B.

Syntax

```
class A
{
   Body of Base class A
};
class B
{
   Body of Base class B
};
class C : public A, public B
{
   Body of derived class C
};
```

Polymorphism —

Polymorphis is another important oops concept. Polymorphis, a greek term, means the ability to take more than one form.

The concept of polymorphism is that "one thing is used in many form"

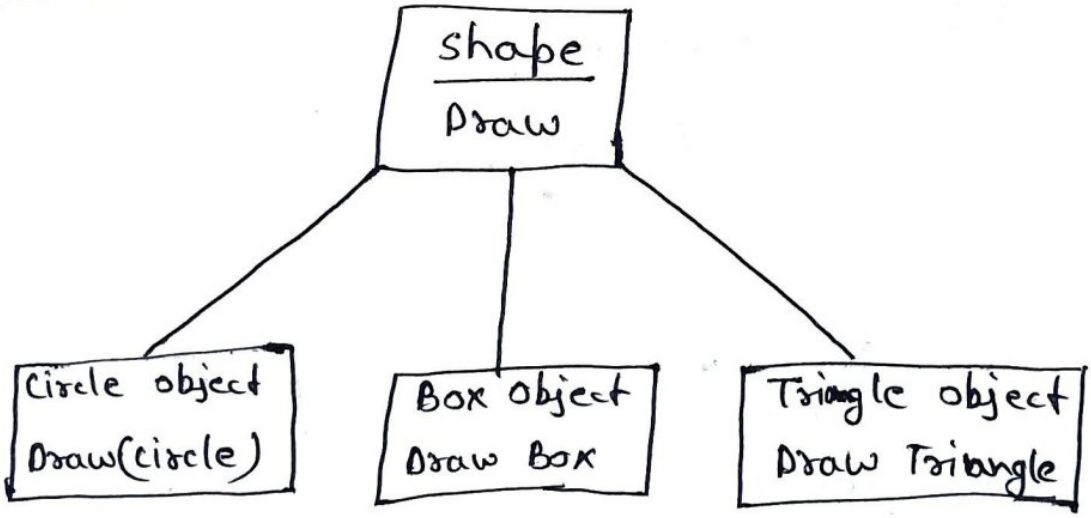
Polymorphism means to process objects differently base on their data types.

Polymorphism cauld be static and dynamic both.

Method overloading is static polymorphism.

Method overring is dynamic Polymorphism.

For Ex



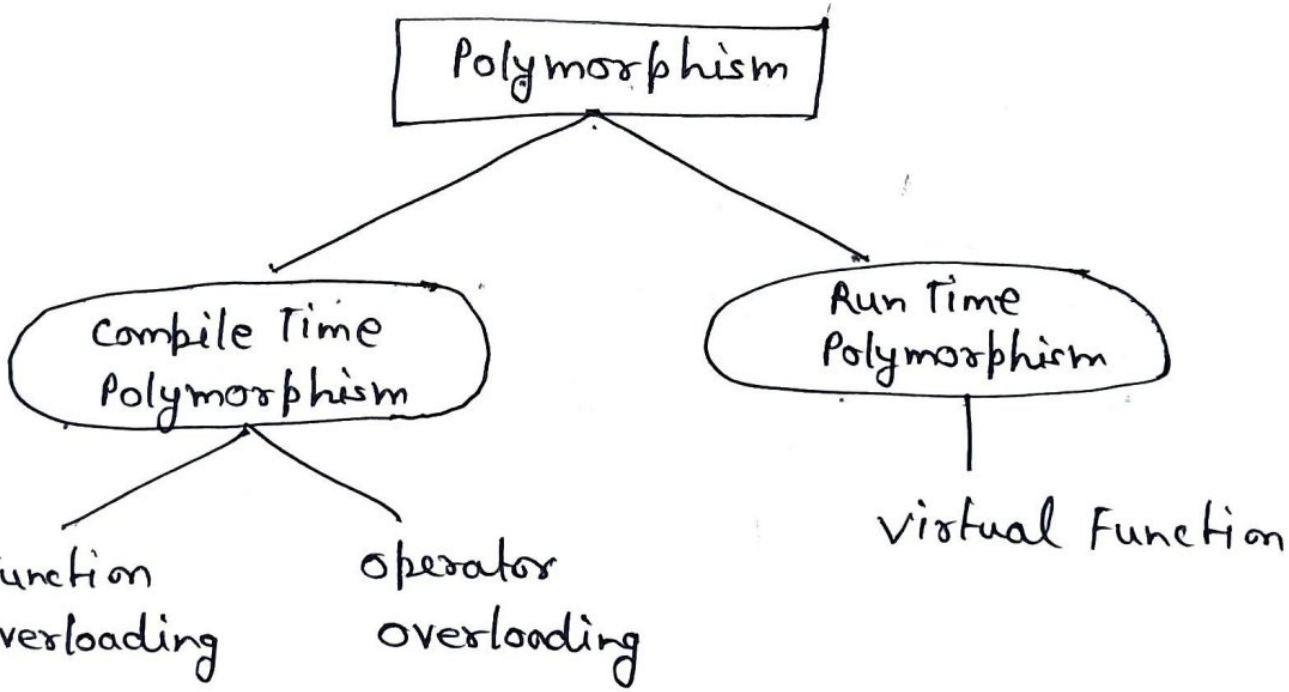
Polymorphism

Types of Polymorphism

There are two type of polymorphism in c++.

1. **Compile Time Polymorphis** – In compile time polymorphism the compiler determines which function is to be executed based on parameters passed to the function or the return type.
It is achived by function overloading and operator overloading which is also known as static binding or early binding.

2. **Runtime Polymorphism** – In Run time Polymorphism the function that is to be choossed for execution is done during execution (run time).
It is also known as latebinding or dynamic binding. It is achived by method overriding and virtual function.



Dynamic Binding —

Dynamic Binding (Dispatch) means that a block of code executed with reference to a procedure (method) call is determind at runtime. Dynamic Binding is also known as late binding or Run time binding Dynamic binding is an object oriented concept and it is related with polymorphism and inheritance.

Message Passing —

objects communicate with one another by sending and receiving information much the same way as people pass message to one another. The concept of message passing makes object easier to talk about binding system.

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

Ex

employee.salary(name)

object — message — information.

Benifit/Advantages of OOPs —

OOPs offers several benifits to both program designer and the user.

1. Through Inheritance, we can eliminate redundant code and extend the use of existing class.
2. The Principle of data hiding helps the programmer to build secure programs that can not be invaded by the code of other programs.
3. It is easy to partition the work in a project based on objects.
4. Object oriented system can be easily upgraded from small to large system.
5. Software complexity can be early managed.
6. New data and function can be early added whenever necessary.
7. With the help of polymorphism, the same function or same operator can be used for different purpose.
8. Emphasis on data rather than procedure.

Object Oriented Modeling —

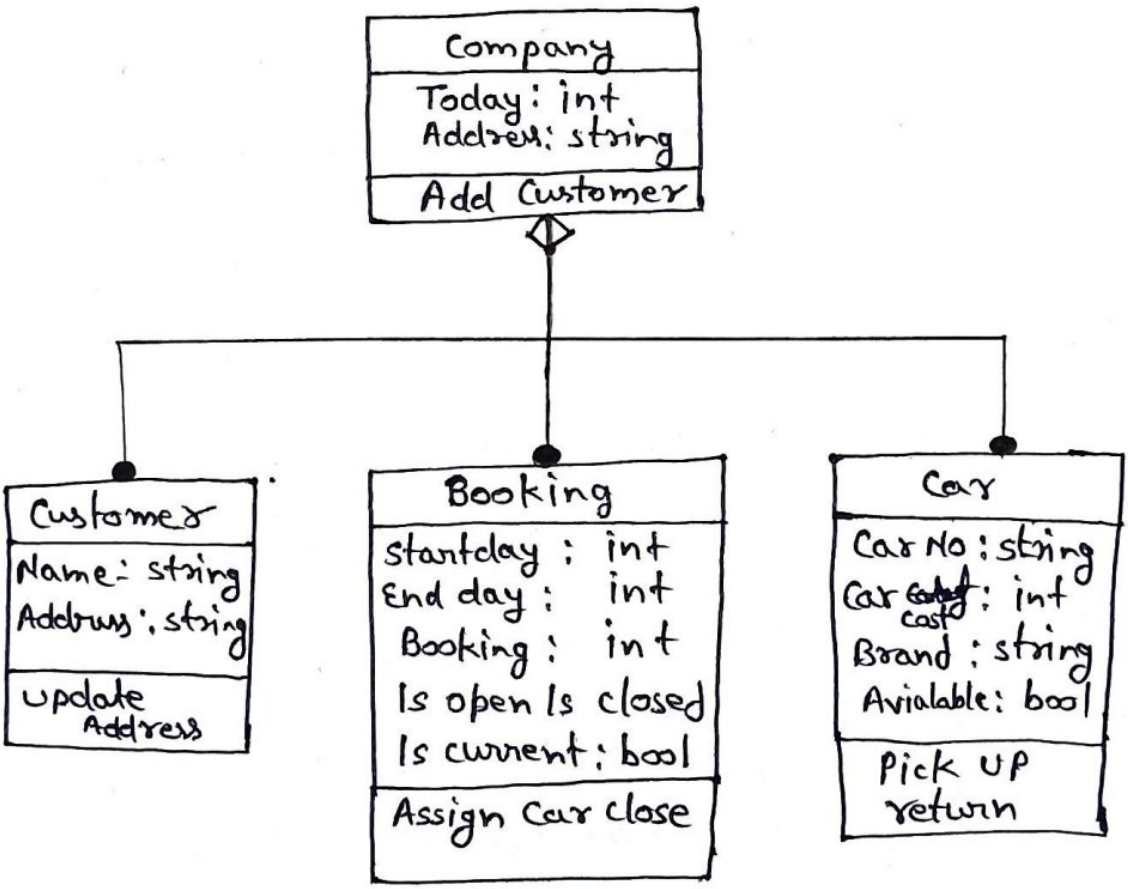
Object Oriented modeling is an approch to modeling an application that that is used at the begning of the software lifecycle when using object oriented approch to software development.

OOM is the construction of objects using a collection of objects that contain stored value of the instance variable found within an objects. object oriented modeling allows for object identification and communi-cation while supporting data abstraction, inheritance and encapsulation.

object oriented modeling is the process of preparing and designing what the model's code will actually look like. During the construction or programming phase, the modeling technique are implemented by using a language that supports object oriented programming model.

Object Model

An object model is a logical interface, software or system that is modeled through the use of object oriented techniques. It enables the creation of an architectural software or system model prior to developement programming.
An object model is a part of object oriented programming life cycle.



Object Model For the Car Rental Company

Benifits of Object Model —

The benifits of using the object Model are —

1. It helps in faster developement of software.
2. It is easy to maintain. suppose a model an error, then a programmer can fix that particular module, while the other parts of the software are still running.
3. It enables reuse of object, design, function
4. It reduce the developement risk.

Dynamic Model

The dynamic model represents the time dependent asbects of a system. It is concerned with the temporal changes in the states of the objects in a system.

The main concepts are.

1. state which is the situation at a perticular condition during the lifetime of a object.
2. Transition, a change in a state.
3. Event, an occurance that triggers transitions
4. Action, an uninterrubted and atomic computation that occurs due to some event.
5. Concurrency of transition.

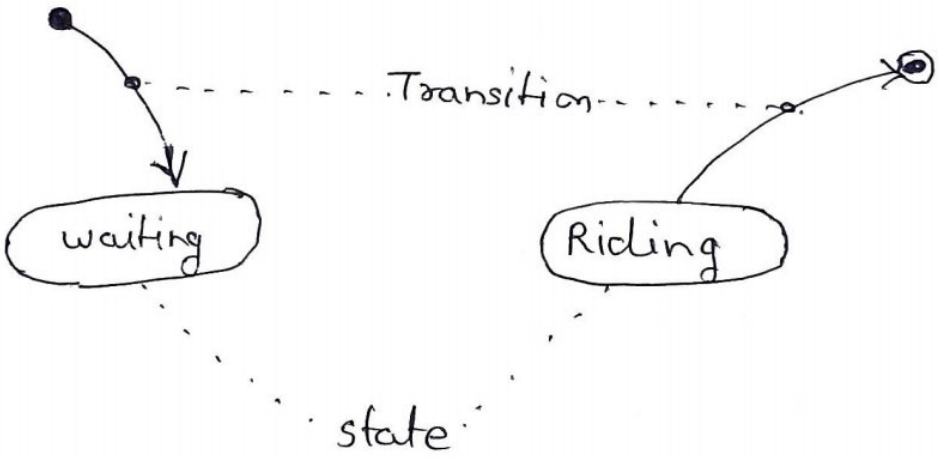
The dynamic model is described with

- state diagram: one state diagram for each class with important dynamic behaviour
- sequence diagram: For intraction between classes

Purpose :- Detect and supply methods for the object model.

For Ex- suppose a person is taking taxi from place x to place y. The state of the person may be: Waiting (waiting for the taxi), Riding (he has got a taxi and is travelling in it), and reached (he has reached destination).

The following figure dipict the state diagram.
↓
transition



State Transition Diagram

Functional Model —

The Functional model shows how values are computed. It shows how output values in a computation are derived from input values, without regard for the order in which the values are computed.

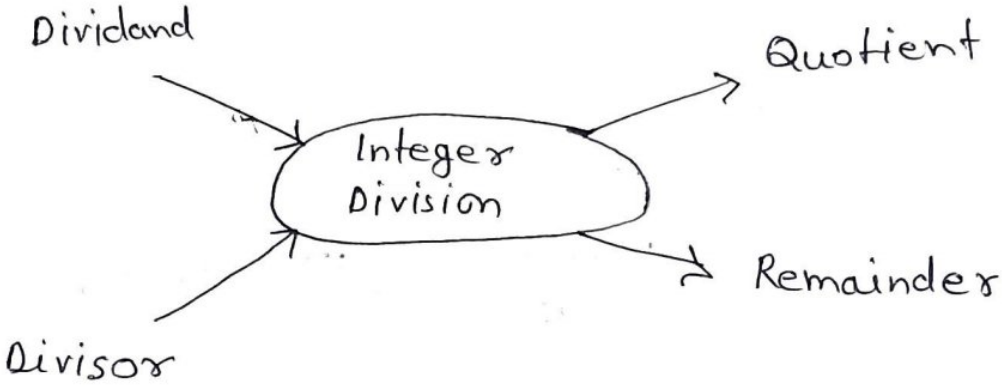
The following steps are performed in constructing a functional model.

1. Identify input and output value.
2. Building DFD showing functional dependencies.
3. Describe function.
4. Identify constraints.
5. specify optimization Criteria.

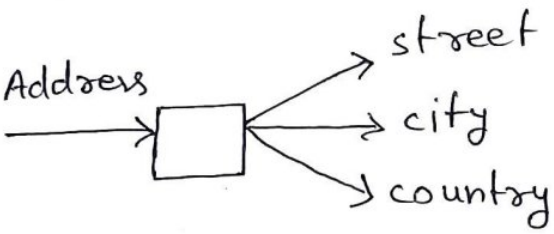
The functional model consist of multiple data flow diagram which specify the meaning of operations and constraints.

A DFD show the functional relationship of values computed by a system, including input values, output values & internal data stored.

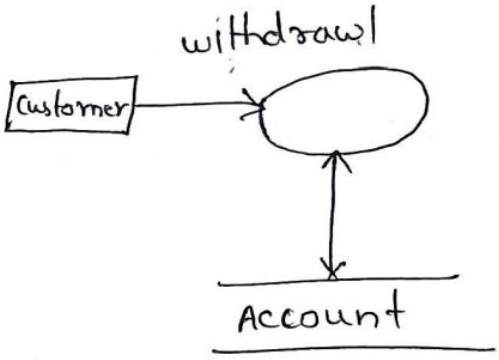
The major component shown in a data flow diagram are shown below.



Process

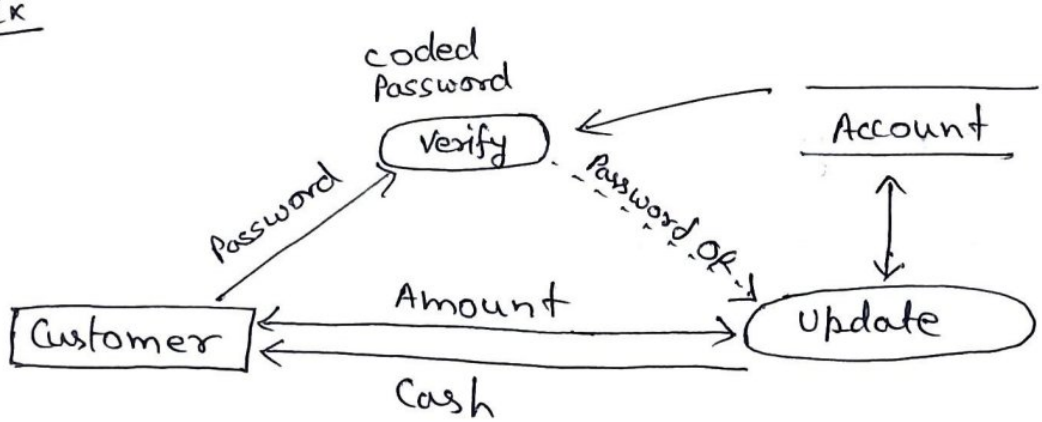


Data Flow



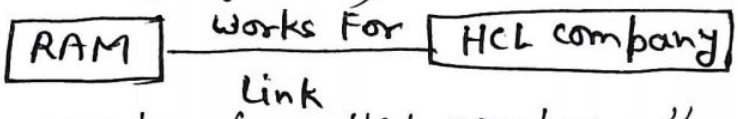
Data store

Ex



Link and Association

Link — Link Provide a relationship between the objects. These objects or instance may be same or different in datastructure & behaviour. Link is a physical & conceptual connection b/w instances (objects).

For Ex — 

" Ram works for HCl company"

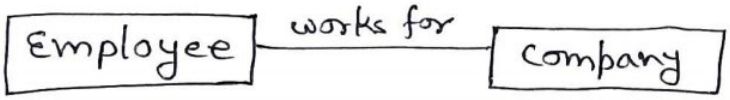
here "works for" is the link b/w Ram and HCl company.

Type of Link —

1. **One to one** — A single object of class is associated with a single object of class B
   Ex- A department has exactly one head and one head can lead only one department.
2. **One to many-many to one** - A single object of class A is associated with many object of class B.
   Ex- A department has many professor.
3. **Many - to Many** — An object of class A may be associated with many object of class B.
   Ex - Many professor teach many student.

Association —

Association is a group of links having comman structure & comman behaviour. The association is the relationship among classes

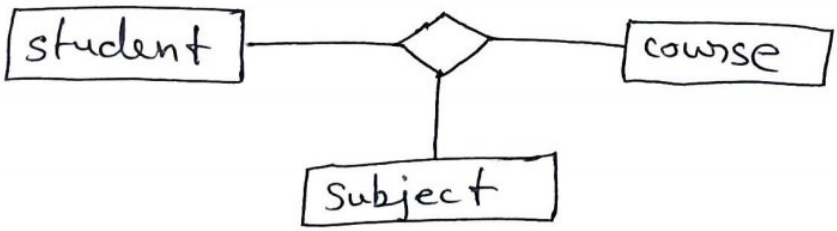


- Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, ternary.

Unary — the association can be defined on a single class.

Binary — The binary association contain degree of two class.

Ternary — The ternary association contain the degree of three classes.



Generalization and specilization —

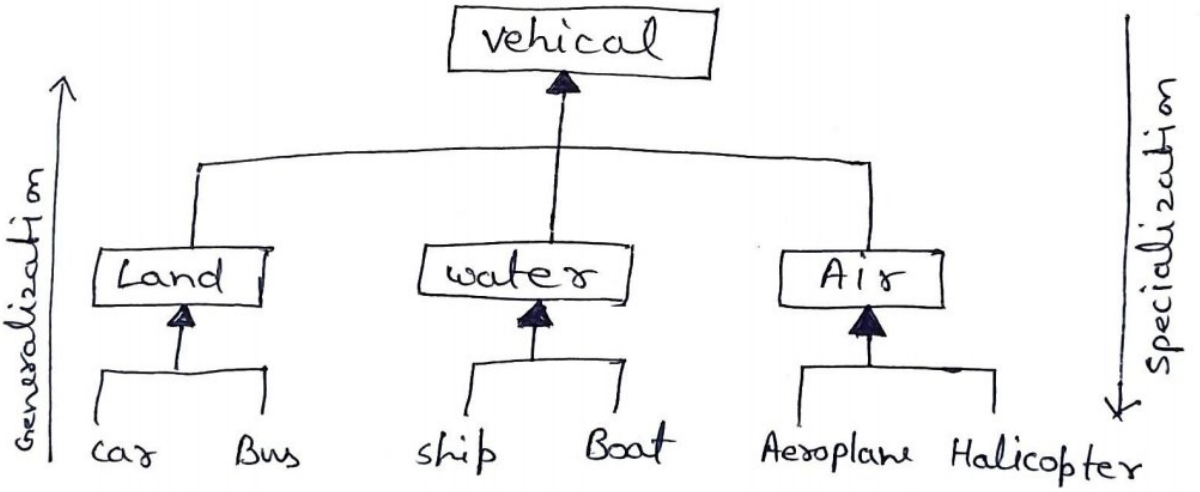
The generalization is a process in which the common characteristics of classes (subclasses) are combined to form a generalised super class. It represents an "is-a kind-of" relationship.

for ex — "Car is-a kind-of land vehical"

OR "ship is-a kind-of water vehical"

Specilization is the reverse process of generalization. It is process of defining specialized subclass from an existing class.

The following figure show the generalization & specialization.



Metadata —

Metadata is "data" that provides information about other data i.e "data about data". Many real world application have metadata such as catalogs, blueprints and dictionaries

For Ex — A text document's metadata may contain information about how long the document is, who the author is, when the document was written, and a short summary of documents.

Object Oriented Design —

Object oriented design is the process of planning a system of interacting objects for the purpose of solving a software problem. It is one approch to software design.

The stages for object oriented design can be identified as.

1. Definition of the context of the system
2. Designing system architecture
3. Identification of object in the system
4. construction of design models
5. specification of object interfaces

oob is divided into two major activities. System design & object design.

System design creates the product architecture, defining a seriese of layers that accomplish specific system functions and identifying the classes that are encapsulated by sub_system that reside at each layer.

1. Object design include the following phases
2. object identification
3. object representation i.e construction of design models
4. classification of operations
5. Algorithm design
6. Implementation of control for external intraction.

Structured Analysis / structured Design

structured Analysis is a set of techniques and graphical tools that allow the analyst to develope a new kind of system specification that are easily understandable to the user This methodology is based on the fallowing principles

1. Top-down decomposition approch.
2. Divide and conquer principle.
3. Graphic representation of analysis result.

There are various tools of SA.

1. Data Dictionary Decision table, Decision Tree DataFlow Diagram.

structuredDesign — structured Design is a methodology to break down a problem in to smaller problems and then salving the smaller problem. structed design is the art of designing the components of a system and their interrela-tionship in the best possible way.

Goals of SASD —

1. Improve the quality and reduce the ~~system~~ risk of system failure
2. Establish concreate requirments specification and complete requisments documentation
   Focus on Reliability, flexibility and maintainability of system.

Introduction to c++ —

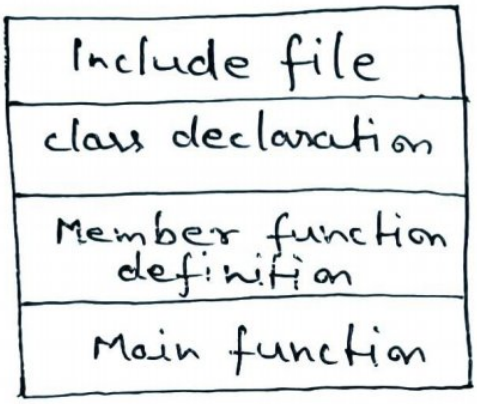
c++ is an object oriente programming language developed by Bjarne Stroustrup at AT&T Bell laboratories in 1980's. c++ runs on a variety of plateform such as windows, Macos, and various version of unix. c++ is a staticaly typed, compiled general purpose, case sensitive, free from programming language that supports procedur object oriented and generic programming. c++ is the increment version of c i.e c with classes. c++ support all the concepts of oops which are given below.

1. Object and class
2. Data Abstraction
3. Encapsulation
4. Inheritance
5. Polymorphism

c++ is a middle-level language, as it comprises a combination of both high level and low level language features.

structure of c++ program

The structure of c++ program is given below.



A Simple c++ program

```
#include<iostream.h>
#include<conio.h>
void main()
{
cout<<"c++ is better than c";
getch();
}
```

preprocessor directives is an instruction to the compiler. The preprocessor directives tells the compiler to include another file into your source file.

iostream file — The <iostream.h> header file which is used for input & output statement

Input/output – In c++ we use new features cout and cin.

cout – cout is used for print the statemer in c++

Ex

```
cout<<"Hello world";
```

The operator << is called insertion or put to operator.

cin – cin is used to take input from keyword.

Ex

```
cin>>a;
```

The operator >> is known as extraction or get from operator.

comment line – In c++ we can have two types of comment.

/* start a paragraph comment which is to terminate by */. All the line and text enclosed b/w there two ~~tiro~~ will be ignored

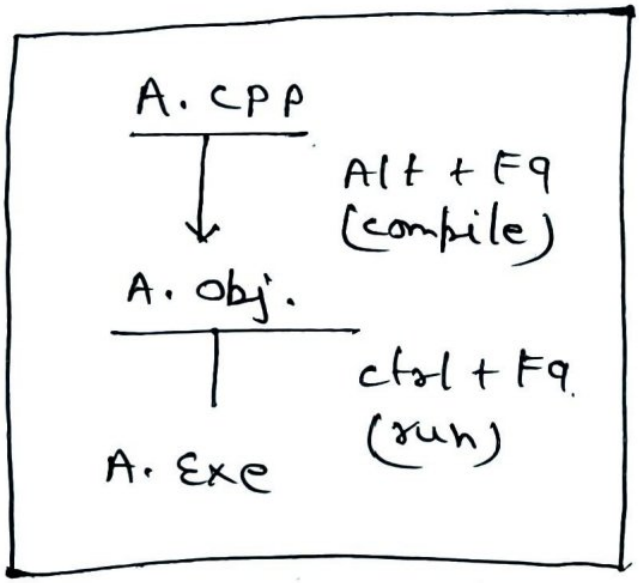
// is used for inline comment.

Compilation and Execution of the program

For Turbo c++ and BORLAND C++

1. First we open the c/c++ Environment then write the c++ code
2. Then we compile the source code with the help of (Alt + F9 key) or compile option from menu the object file is created.
3. Then we link the object file (ctrl + F9) then the direct executable file (.Exe) is created.

For Ex



Keywords — There are many words predefined by language and which have some special meaning is known as keywords. There are 60 keywords in c++. c++ is a case senstive language and it requires that all keywords be in lower case. Some keywords are —

asm	double
auto	else
break	enum
case	extern
catch	float
char	for
class	

Variables — variables are means for location in memory used by a program to store data variable names may have different datatype to identify the type of value stored.

for ex — int a = 5; // declared and initialized here a is a integer type variable which hold the value 5.

2

Scope of variable

All the variable have their area of functioning, and out of that boundary they do not hold their value, this boundary is called scope of variable.

scope of variable are of two type

1. Local variable — Local variable are the variables which exist only between the curly braces, or inside the main() function

```
void main()
{ int n=10;
  if (i<20) // if condition scope start
  { int n=100;
  }          // if condition scope ends
  cout<< n; // compile time error n not
              avilable here.
```

2. Global variable — Global variable are those, which are once declared and can be used throughout the lifetime of program by any class of or any function. They must be declared outside the main()

```
int x;
void main()
{
  x = 10;  // initialized once
  cout<<x;
  x = 20;  // initialized again
  cout<<x
}
```

Constant constant are also normal variables but only difference is, their values can not be modified by the program once they are defined. constant refers to the fixed value They are also called as literals.
constant may belonging to any data types.

Ex

[const datatype variable name ;]

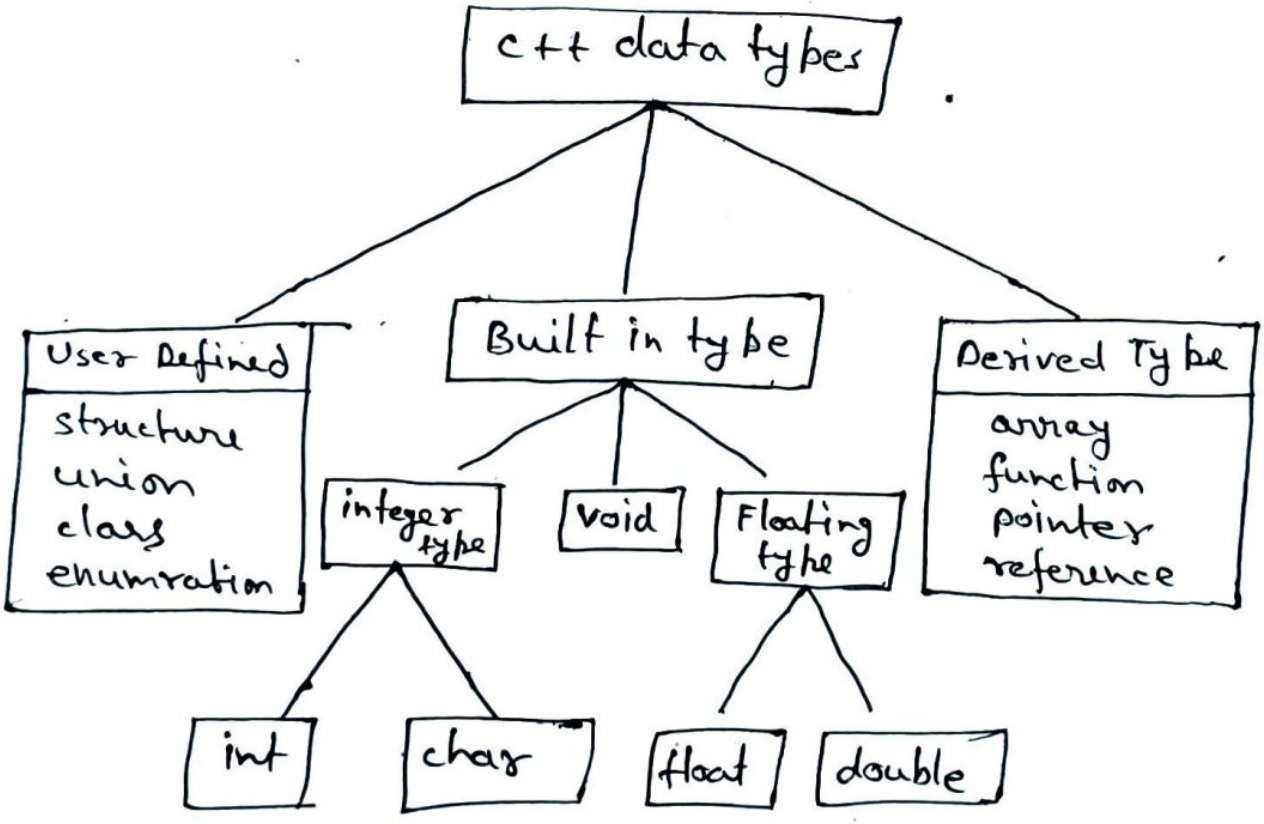
Type of constant

integer constant, real or floating point constant, string constant etc character constant.

Data types

Data types are the predefined keywords which are used to allocate the memory according to the user's requirment. Datatype defines the values that a variable can take. Datatypes in c++ can be classified under various categories as shown in figure.

Ex: If a variable has int data type it can only take integer value.



User defined data type —

We have used user-defined datatype such as struct and union in c. In c++ some more features have been added. c++ also permit us to define another user defined data type known as class. we can define the variable of class which is known as objects.

Enumerated Data types —

Enumerated data type is user defined datatype which provides a way for attaching name to number, so increasing comprehensibility of the code. The enum keywords automatically enumerated a list of words by assigning them values 0, 1, 2 and so on. The syntax of enum is similar to struct statement.

```
enum shape { circle, square, tringle }
enum colour { red, blue, green, yellow }
```

by default the enumerator are assigned integer value starting with 0 for the first enumerator, 1 for second, and so on.

```
enum colour { red, blue = 4, green = 8 };
enum colour { red = 5, blue, green }
```

then for 1 statement red = 0
" for 2 statement blue = 6, green = 7

Program of Enum

```
#include<iostream.h>
#include<conio.h>
enum week
 { Monday, Tuesday, Wednesday, thursday,
   Friday = 8, saturday, sunday };
void main()
{
  clrscr();
  week day;
  day = Friday;
 cout << " Day = " << day + 1 << endl;
 cout << " value of Monday = " << Monday << endl;
 cout << " Tuesday " ;
 cout << " Thursday " ;
 cout << " Friday " ;
 cout << " saturday " ;
 sout << " sunday " ;
 getch();
}
```

output

```
Day = 9
value of Monday = 0
      Tuesday   = 1
      Thursday  = 3
      Friday    = 8
      Saturday  = 9
      sunday    = 10
```

structure

structure is a callection of different data tybe and having a common name.
Data member of structure can not be directly access and it can be directly access with the help of structure variable and . (dot) operator.

Syntax of structure is

```
struct <structure name>
{
   <data members> ;
} <structure variable> ;
```

Example

```
struct student
{
   int id, age ;
   char name [25];
} s;
```

Here s is a structure variable and 29 bytes memory is allocated for the above structure.

Program to display student record using structure

Sol

```
struct student
{
   int id, age;
   char name[25];
}s;
void main()
{
  clrscr()
cout<<"\n Enter the student id";
cin >> s.id;
cout<< "\n Enter the student age";
cin >> s.age;
cout<<"\n Enter the student name";
cin << s.name;
cout<<"student id is:"<< s.id;
cout<<"student age is ="<<s.age;
cout<< "student name is ="<< s.name;
getch();
}
```

output

```
student id is = 101
student age is = 25
student name is = Akash
```

union –

A union is a user defined data type in which all members share the same memory location. This means that at any given time a union can contain no more than one object from its list of members. Union is also similar to the structure syntax is also same. But the difference between structure and union is in term of memory storage location.

In structure all member has its own memory storage location while in union all the member shared same memory location. The size of union is equal to size of largest data type.

Syntax of union

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

Write a Program to differentiate structure and union.

```
#include<string.h>

union Data{
        int i;
        float f;
        char str[20];
    };
struct Data1 {
        int i;
        float f;
        char str[20];
    };
void main()
    {
     clrscr();
     union Data data;
     struct Data1 data1;
     data.i = 10;
     data.f = 220.5;
     strcpy(data.str, "C++ Programming");
     cout<<"Example of union"<<endl;
     cout<<"size of union = "<<sizeof(data)<<endl;
     cout<< data.i<<endl;
     cout<< data.f<< endl;
     cout<< data.str<<endl;
     data1.i = 10;
     data1.f = 220.5;
     strcpy(data1.str, "C++ Programming");
```

```
cout<<"Example of structure"<<endl;
cout<<"size of structure ="<<sizeof(data1)<<endl;
cout<<data1.i<<endl;
cout<<data1.f<<endl;
cout<<data1.str<<endl;
getch();
}
```

output

```
Example of union
size of union = 20
11075
1.449858e-19
c++ programming

Example of structure
size of structure = 26
10
220.5
c++ programming
```

Operators —

operators are special type of character or symbol in c++ that performs operations on arguments to return a specific result.

Types of operators —

1. Arithmatic operator
2. Relational operator
3. logical operator
4. Increment Decrement operator
5. Assignment operator
6. comparison operator
7. Conditional operator (Ternary)
8. Bitwise operator

Ternary operator —

Ternary operator deals with three operands. It is also called conditional assignment statement because the value assigned to a variable depends upon a logical expression.

syntax –

variable = (test expression) ? expression1 : expressio

```
a = 5, b = 3;
max = (a > b) ? a : b;
```

here value 5 is stored in max because a>b is true.

Bitwise operator – In c++ we can use some special types of operators, which perform operations on Bit level of the operands.

Symbol	Operation	Example
&	Bitwise AND	x & y
|	Bitwise OR	x | R
<<	shift Left	x << 2
>>	shift Right	x >> 2
~	Bitwise not	~x

These operators use byte, short int long type operands.

Precedence of operator –

Precedence of operat. is used to determine the order in which different operators in a complex expressior are evaluated. The operators at the higher levels of precence are evaluated first. The operators of some precedence are evalu either from left to right or from right to left depending on the level. This is known as "the associativity property of a operator"

operator	Precedence	Associativity
( ), [ ]	1	L to R
++ -- ~	2	R to L
*, /, %	3	L to R
+, -	4	L to R
<<, >>	5	L to R
<, <=, >, >=	6	L to R
&, ^, !	7	L to R
&&, ||	8	L to R
,	9	L to R

Some new operators in c++ —

c++ introduce some new operators.

:: - scope resolution operator.
new - memory allocation operator.
delete - memory release operator.
endl - Linefeed operator.
setw - Field width operator.

scope resolution operator —

In c language if we define the name of local and global variable are same, and we want to access the global variable within main() then it is not possible but in c++ it is possible with the help of scope resolution operator. The major application of scope resolution operator is in the class to identify the class to which a member function belongs.

ex—

```
int c = 20;
void main()
{
  c = 25;
  cout<< c;        // 25
  cout<< ::c;      // 20
  getch();
}
```

new —

It is memory allocation operator

```
P = new int ;
```

memory is allocated for P integer variable if sufficient memory is not available for allocation, new operator return null pointer

Delete —

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

```
delete object a ;
```

setw —

It is field width operator

```
cout << setw(4) << sum << endl ;
```

The manipulator setw(4) specifies a field width 4 for printing the value of the variable sum.

Difference between C & C++

	C	C++
1	C is a procedural programming language and does not support class & objects	while C++ is a combination of both procedural & object oriented programming language
2.	C is a subset of C++	C++ is a super set of C.
3	C does not support the concept of oops	C++ support the concept of oops.
4.	C does not support function and operator loading	C++ support the function & operator overloading
5.	C use function for input/output scanf and printf	C++ use object for input /output for ex- cin, cout
6.	Top down approach is used in program design	Bottam up approach adopted in program design.

Storage classes — A storage class defines the scope visibility and life-time of variables and/or functions within a c++ program.
How storage is allocated for variable and how variable is treated by compiler depends on these storage class. There are following storag classes.

1. Auto
2. Register
3. static
4. extern
5. mutable

1. Auto — This is default storage class. All variables declared are of type auto by default.
A variable declared inside a function without any storage class specification is by default an automatic variable. In order to declaration of variable we auto keyword.

Syntax — `auto int num;`

The lifetime of auto variable exists as long as controls remains in the block and function.
Its default initial value is Garbage collection

2. Register storage classes —

The variables of class Register are stored in CPU Register instead of memory which allows faster access. Register variable are similar to auto variable and exists inside that perticular function only The default value is Garbage value.

Syntax —

```
register int count;
```

3. static —

static variable have a special features that they retain their value during the excution of program. The default value of static variables is garbage. The scope of static variable can be local or global. lifetime of static variables is equal to lifetime of programs.

syntax —

```
static int num
```

3 extern — variables declared outside all functions (including main()) comes under the external storage class. The scope of extern varia is global. Lifetime of external variables is equal to lifetime of program. When you have multiple files and you define a global variable or function, which will be used in other file also, then extern will be used in another file to give reference of define variab or function. An extern variable can be shared accross multiple files.

syntax- `extern int num`

4. mutable storage — The mutable storage class specifiers is used only on a class data membe to make it modifiable even through the mem is part of an object declared as const. You can not use the mutable specifiers with name declared as static or const or referenc member.

syntax—

```
class test
{
mutable int x;
- - - - -
- - - -
};
```

Example of Auto storage class

```
void demo()
{
  auto int num = 10;
  cout<< num;
}
int main()
{  demo();
   cout<< num;      // Error. Undefined Identifier.
   return 0;
}
```

Example of Register storage class.

```
void demo()
{
  register int num = 10
  cout<< num;
}
void main()
{
   demo();
   return 0;
}
```

Example of Extern storage class

```
int a=100;
void demo()
{
  extern int a;
   a=200;
  cout<< a;
int main()
{  demo();
  return 0;
}
```

Example of static storage class

```
void count()
{
   static int num=0;
   num++;
   cout<<num;
}
 int main()
 {
    count();
    count();
    cout();
  return 0;
 }
```

output

```
1
2
3
```

without static output

```
1
1
1
```

Example of mutable storage class.

```
class demo
{
  public:
    mutable int x;
             int y;
};
int main()
{
 const demo d={5,10};
   d.x = 100;
 cout<<"x="<<d.x;
 cout<<"y="<<d.y;
 return 0;
}
```

output

```
x= 100
y = 10
```