UNIT - 1

(1.) Difference between POP and OOP (or) C and C++ :

POP (or) C	OOP (or) C++
1. POP stands for Procedure oriented Programming.	1. OOP stands for object oriented Programming.
2. It is a TOP down approach.	2. It is Bottom-up approach.
3. POP gives more importance to functions.	3. OOP concentrates more on the data and treats data as critical.
4. In this there is no data abstraction & Encapsulation.	4. Here data Security is provided using keywords ~~for~~, access specifiers Private, Public, Protected.
5. There is no concept of reusability in POP.	5. Reusability is Provided by using the inheritance concept.
6. Data flows freely around the system from function to function.	6. Objects can move and communicate with each other through member functions.
7. large Programs are divided in to smaller Programs known as functions.	7. Programs are divided into what are known as objects.
8. It is not so easy to add new ~~add~~ data and function.	8. Provides an easy way to add new data and function.

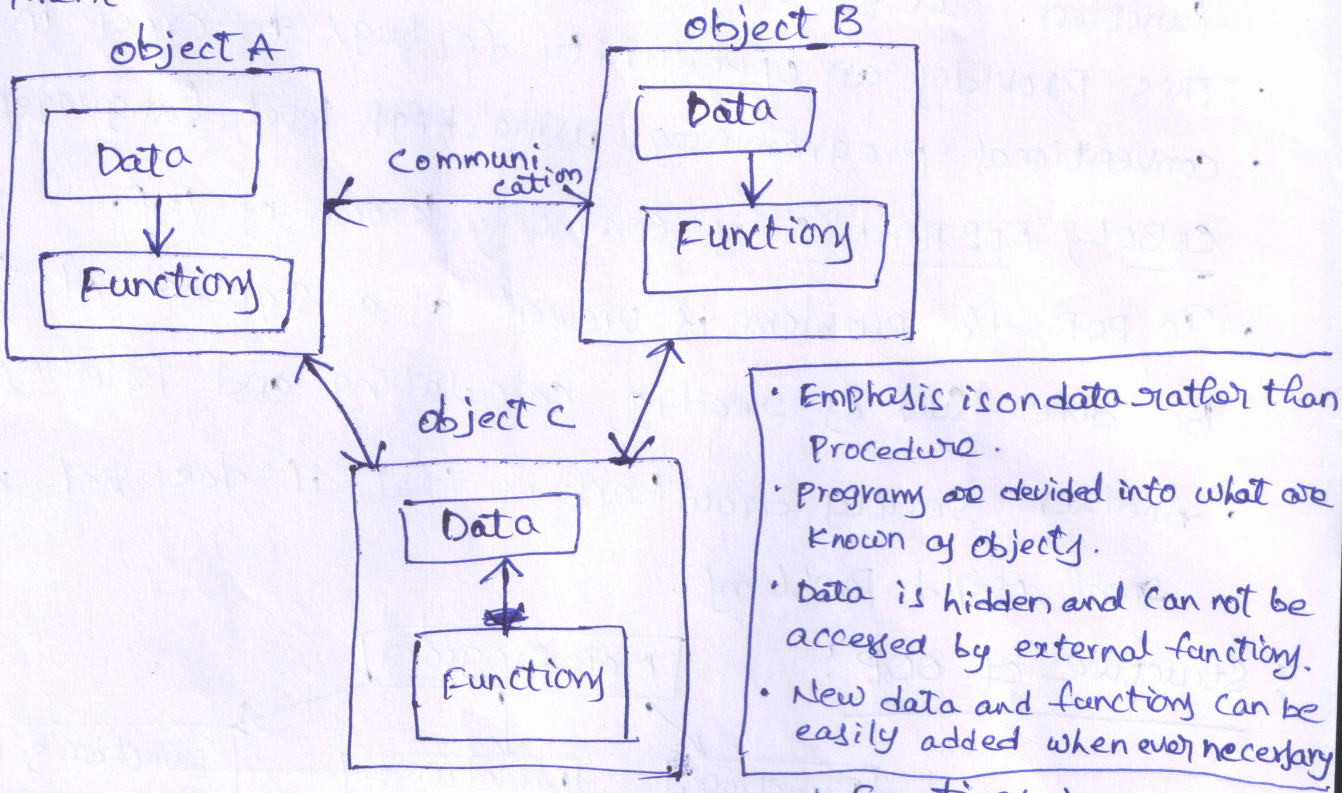
(2.) Benefits or Advantages of OOP:

OOP offers several benefits to Program designer and the users.

* Reusability: In OOP's Programs functions and modules that are written by a user can be reused by other users without any modification.
* Inheritance: Through this we can eliminate redundant code and extend the use of existing classes.
* Data Hiding: The Programmer can hide the data and functions in a class from other classes. It helps the programmer to build the secure Programs.
* Reduced Complexity of a Problem: The given Problem can be viewed as a collection of different objects. Each object is responsible for a specific task. This technique reduces the complexity of the Program design.
* Easy to maintain and upgrade: OOP makes it easy to maintain and modify existing code as new objects can be created with small difference to existing ones.
* Message Passing: The technique of message communication between objects makes the interface with external systems easier.
* Modifiability: It is easy to make minor changes in the data representation or the Procedures in an OO program. changes inside a class do not affect any other part of a Program.

→ write about Object oriented Programming :

- It Fallows bottom-up approach in program design.
- object-oriented Programming is a Programming Paradigm that uses abstraction (in the form of classes and objects) to create models based on the real world environment.
- An object-oriented application uses a collection of objects, which communicate by passing messages to request services.
- Objects are capable of passing message, receiving messages and processing data.
- The aim of object-oriented programming is to try to increase the flexibility and maintainability of programs.
- Because Programs created using an OO language are modular, they can be easier to develop, and simpler to understand after development.



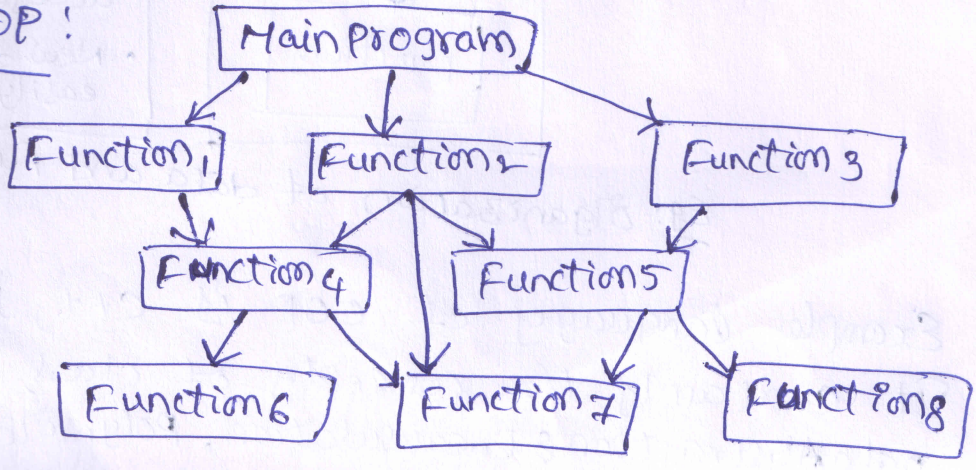
Eg: Organization of data and Functions in OOP.

- Example languages of OOP is C++, java etc.
- It contains the concepts of class, object, inheritance, Data Abstraction & Encapsulation, Polymorphism, Dynamic binding and Message passing.

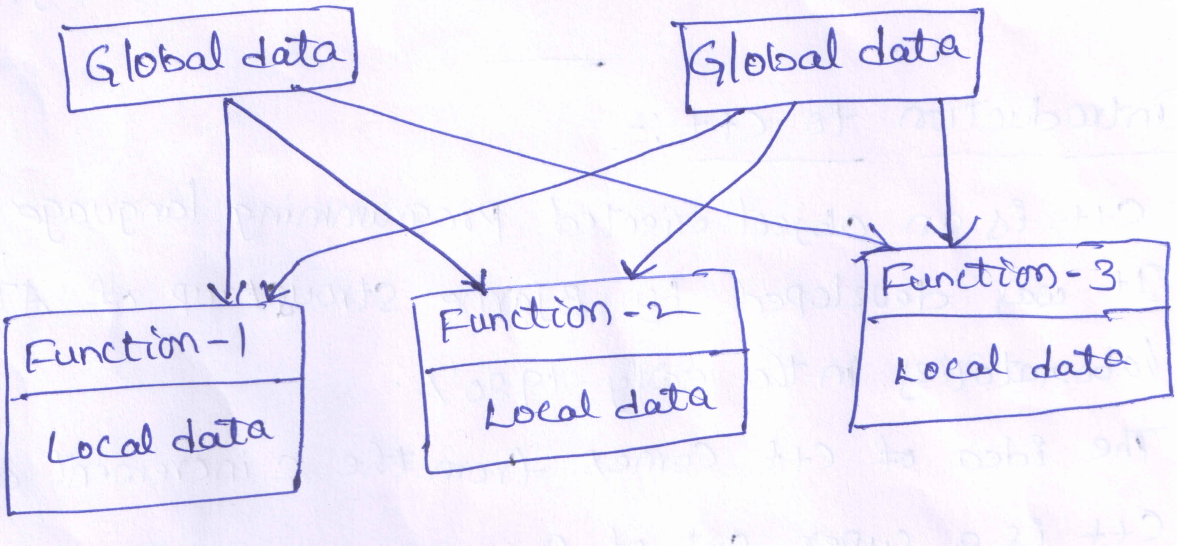
→ write about procedure oriented Programming:

- All computer programs consists of 2 elements.
  1. code or function 2. Data.
- In general a program can be organized in one of the 2 ways
  1. Around its code 2. Around its data.
- In POP, programs are organized around code (function)
- This Approach can be treated as code acting on data.
- In POP, we concentrate mainly on the development of functions.
- A very little attention is given to the data.
- In POP Data move openly around the system from function to function.
- It is very difficult to identify what data is used by which function. It is top-down approach.
- This provides an opportunity for bugs to creep in.
- conventional programming using high level languages such as COBOL, FORTRAN, C is commonly known as POP.
- In POP the problem is viewed as a sequence of things to be done such as reading, calculating and Printing.
- Another serious drawback is that it does not model real world problems.

: Structure of OOP:



• Relationship of data and functions in Procedural Programming



→ Disadvantage of conventional Programming:

- Traditional Programming languages such as COBOL, FORTRAN etc. are commonly known as Procedure oriented language.
- The Program written in these languages consists of a sequence of instructions that tells the compiler to perform a given task.
- when program code is large then it becomes inconvenient to manage it.
- To overcome this Problem, Procedures or Subroutines were used
- A Program is divided into many functions. Each function can call another function.
- If the Program is too large the functions also creates Problems.
- In many Programs, important data variables are declared as global
- In case of Programs containing several functions, every function can access the global data.

to understand the various data used in different functions.

→ As a result, a program may have several logic errors.

⇒ Introduction to c++ :-

- c++ is an object oriented programming language.
- It was developed by Bjarne Stroustrup of AT&T Bell laboratories in the early 1980's.
- The idea of c++ comes from the c increment operator ++.
- c++ is a super set of c.
- c++ is an extension of c with a major addition of the class, inheritance, Polymorphism features.

→ Applications of c++ :-

- c++ is a versatily language for handling very large Programs
- It is suitable for developing of editors, compilers, databases etc.
- c++ Programs are easily maintainable and expandable.
- By using c++ we can build special object oriented libraries which can be used by many Programmers.
- we can also build any complex real-life application systems.

→ Comments in c++ :-

- Comments are used for better understanding of the Program.
- c++ uses 2 types of comments symbols.
  1. // doubleslash 2. /* slash star.
- The double slash comments are referred to as c++ style comments.
- This comment tells the compiler to ignore everything that follows //

Eg: ① // c++ is better than c ② /* c++ is OOP */

Basic concepts of object Oriented Programming:

1. class
2. Object
3. Data Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism
7. Data binding
8. Message Passing.

1. class:

- A class is a user defined datatype.
- It is a collection of variables called data members and functions called member functions.
- class is a way to bind the ~~me~~ data members and member functions with a particular class name.
- The data members and member functions are called as members of a class.
- A class is similar to a structure in C.
- A class is a collection of objects of similar type.

Syntax:

```
class classname
{
    private:
        data members;
        member functions;
    Public:
        data members;
        member functions;
};
```

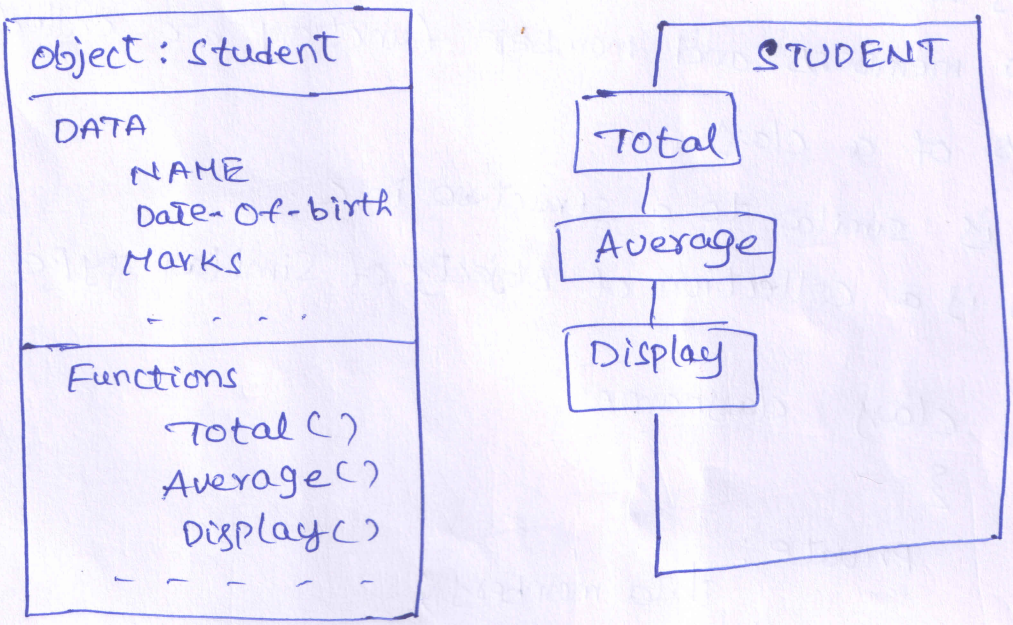
Eg:

```
class Student
{
    Private:
        int Rno;
        char name[20];
    Public: void get_Rno();
```

2. Object:

- objects are instances of a class or class variables.
- once a class has been defined we can create any number of objects related to that class.
- objects are basic runtime entities in an object oriented system.
- An object may represent a person, a place, a bank account or any item that a program has to handle.
- when a program is executed, the objects interact by sending messages to one another.

Two ways of Representing object.



- Syntax of an object is
  - classname objectname;

Eg: Student S1;

3. Data Abstraction:

- Abstraction is a Process of representing essential features without including the background details.
- classes use the concept of abstraction.
- Since the class use the concept of data abstraction. They are called abstract data type (ADT).

4. Encapsulation:

- Data encapsulation is the most striking feature of OOP.
- The wrapping up of data and functions into a single unit is called as encapsulation.
- Encapsulation is a mechanism that binds the data and its associated code together.
- It also protects the data from outside world or modificatio

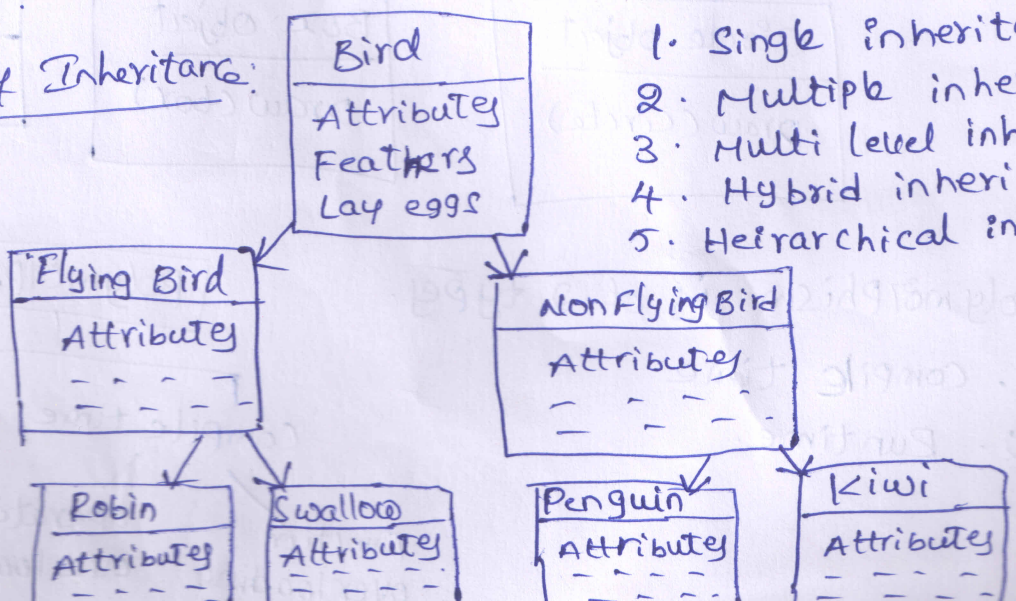
5. Inheritance:

- The concept of inheritance provides the idea of Reusability.
- Inheritance is the process of deriving a new class from an old class.
- The old class is called base class (or) Parent class.
- The ~~derived~~ ~~call~~ new class is called derived class (or) child class.
- The derived class inherits some or all the properties of a base class.
- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- This means that we can add additional features to an existing class without modifying it.
- The new class will have the combined features of both the classes.

There are 5 types.

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

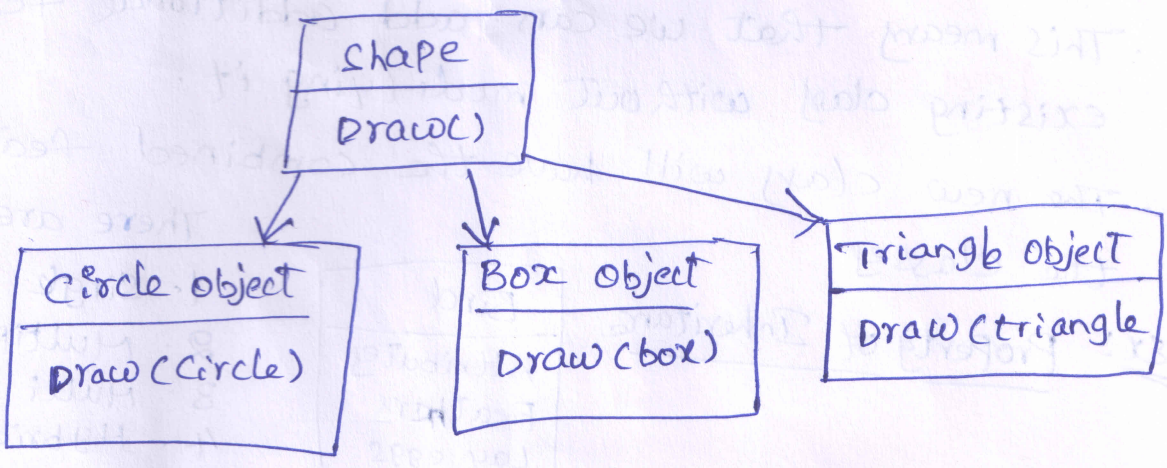
Ex: Property of Inheritance:



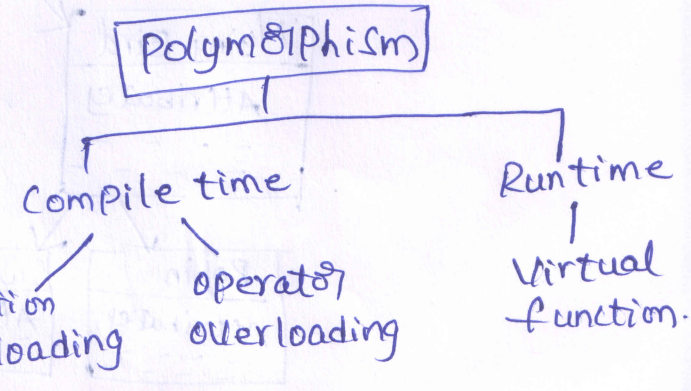
6. Polymorphism:

- Polymorphism is an important concept of OOP.
- The word Polymorphism is a greek word.
- 'Poly' means many. 'Morphism' means forms.
- Polymorphisms means the ability to take more than one form.
- An operation may exhibit different behaviors in different instances.
- The behavior depends on the types of data used in the operation.
- The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.
- Using a single function name to perform different types of tasks is known as function overloading.
- Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface.

Ex:



- Polymorphism is of 2 types.
  1. Compile time
  2. Runtime.



7. Data Binding or dynamic binding:

- Binding means linking the Procedure call to the associated code to be executed.
- Dynamic binding is also known as late binding
- It is associated with Polymorphism and inheritance.

8. Message Passing:

- An OOP consists of set of objects.
- The objects communicate with each other by sending and receiving information.
- It is similar to the way as people pass message to one another.
- Message passing involves specifying the name of object, the name of the function and information to be sent.
- The Basic steps in OO language are
  i, Creating classes that define objects and their behavior
  ii, Creating objects from class definitions, and
  iii, Establishing communication among objects.
- Message Passing involves specifying the name of the object, the name of the function (message) and the information to be sent.

Example:

```
employee . salary (name);
```

object — message — information

Ex: 2

```
Student S;
S.getdata();
```

→ DisAdvantage of Conventional Programming (POP) :-

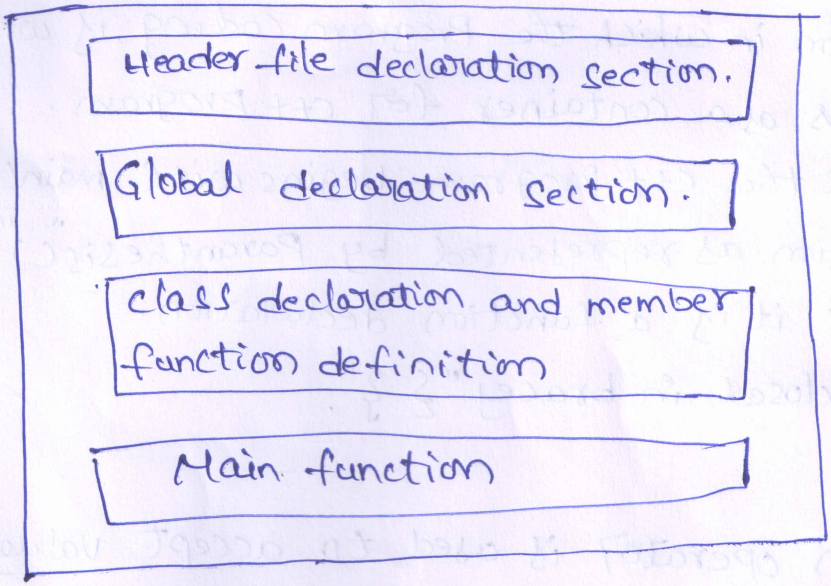
- Traditional Programming language such as COBOL, FORTRAN etc. are commonly known as Procedure oriented language.
- The program written in these language consists of a sequence of instruction that tells the compiler to perform a given task.
- When Program code is large then it becomes inconvenient to manage it.
- To overcome this problem, Procedure or Subroutine were used.
- A Program is divided into many functions. Each function can call another function.
- If the Program is too large the function also creates Problems.
- In many Programs, important data variable are declared as global.
- In case of Programs containing several functions, every function can access the global data.
- Generally in a Program of large size, it becomes difficult to understand the various data used in different functions.
- As a result, a Program may have several logical errors.

→ Structure of C++ Program:-

A typical C++ Program would contain four basic sections.

1. Header file declaration section,
2. Global declaration section.
3. class declaration and member function definition.
4. Main function.

Structure of C++ Program



1. **Header file section:-**

- The #include<iostream> represents header files, which include the predefined functions.
- They are known as the Pre Processor directives.
- The '#' symbol tell about "address to" or "link to" the specified header file.
- The in-built functions such as cout, <<, cin >> etc are linked with header file <iostream>.

Global declaration section:

- There are certain programs which require variables that can be used in more than one function.
- So these variables can be declared out side the main() function or respective functions.
- Then those variables become accessible in any of the functions

Class declaration and member function definition:-

- Class declaration and all the member function definitions are written in this section.

main() section:

- This is the section in which the Program coding is written.
- Basically, it acts as a container for c++ program.
- The execution of the c++ program begins with main() function.
- main() is a function as represented by Paranthesis "()". This is ~~a function~~ because it is a function declaration.
- The body is enclosed in bracey "{ }".

Input in c++:

- In c++ Extraction operator is used to accept values from the user.
- cin is used for accepting data from keyboard.
- The operator >> is known as the extraction operator or get from operator.
- It is similar to scanf() in C.
- Syntax: Cin >> variable;

  Eg: cin>>a; (or) cin>>a>>b>>c;

Output in c++:

- In c++ Insertion operator is used to display value to the user.
- cout is used for displaying data on the screen.
- The operator << is called as the insertion operator or the put to operator.
- It is similar to the printf() statement in c
- Syntax: cout<< "Text to be displayed as output";

  eg: cout<<" enter a and b values";

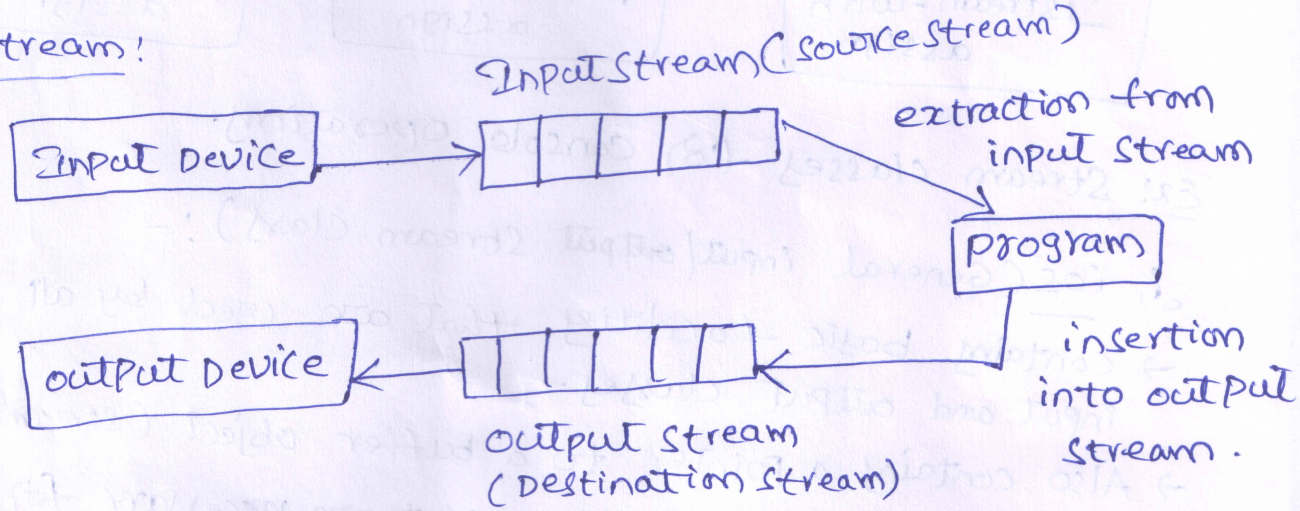
Simple c++ program:

```
#include<iostream>
using namespace std;
int main()
{
    cout<<" c++ is better than c:\n";
    return 0;
}
```

Streams in C++:

- Every program takes some data as input and generates processed data as output, following the input-output process cycle.
- C++ supports a rich variety of C++ I/O functions and operators.
- C++ uses the concept of stream and stream classes to implement its I/O operations, with the console (keyboard) and disk files.
- A stream is a sequence of bytes.
- It acts as a source from which the input data can be obtained or as a destination to which the output can be stored.
- The source stream that provides data to the program is called as the input stream.
- The destination stream receives the output from the program is called as the output stream.

Data Stream:

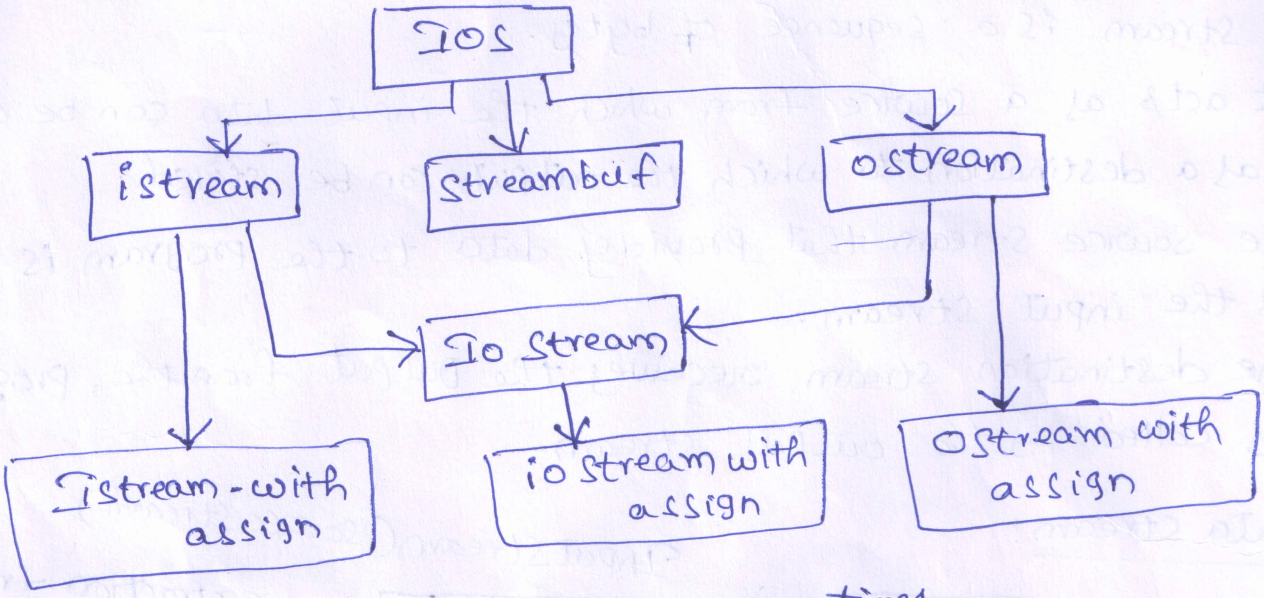


- The data in the input stream can come from the keyboard or any other storage device.
- Similary the data in the output stream can come from the keyboard or any other storage device.
- C++ contains several Pre defined streams, that are opened automatically opened when a program begins its execution.

→ C++ Stream classes:

- The C++ I/O stream contains a hierarchy of class streams that are used to define various streams.
- The classes are called stream classes.

i, Stream classes for console I/O operation:



Ex: Stream classes for console operations.

i, ios (General input/output stream class) :-
- → Contains basic facilities that are used by all other input and output classes.
- → Also contains a pointer to a buffer object (streambuf object)
- → Declares constants and functions that are necessary for handling formatted input and output operations.

ii, istream (input stream) :
- Inherits the properties of ios.
- Declares input functions such as get(), getline() and read().
- Contains overloaded extraction operator >>

iii, ostream (output stream) :
- Inherits the properties of ios
- Declares output functions put() and write()
- Contains overloaded insertion operator <<

iv) iostream (input / output stream):

- Inherits the properties of ios stream and ostream through multiple inheritance and thus contains all the input and output functions.

v. streambuf:

- Provides an interface to physical devices through buffers.
- Acts as a base for filebuf class used ios files.

⇒ Pre defined streams and stream classes:

- C++ contain several streams that are opened automatically when the execution of a program.
- The most prominent pre-defined streams in C++ are related to the consol device.
- These are opened automatically before the function main is executed and closed after the main function has completed.
- These pre-defined stream objects are declared in iostream.h they have the following meaning.
- cin - standard input corresponding to stdin in C
- cout - standard output corresponding to std out in C
- cerr - standard error output corresponding to stderr in C

⇒ Unformatted i/o operations :-

- These are 3 types.

1. overloaded operators >> and <<
2. put() and get() functions.
3. getline() and write() functions.

1. overloaded operators >> and <<:

- we have used the objects cin and cout (pre-defined in the iostream file) for the input and output of data of various types.

- The >> operator is overloaded in the istream class
- The << operator is overloaded in the ostream class.
- The following are the general format for reading and writing data from the keyboard:

i. Syntax for writing of data.

cout << " enter the text or variables";

ii. Syntax for reading of data.

cin >> variable$_1$ >> variable$_2$ ---- >> variable$_N$.

2. put() and get() Functions:

- The classes istream and ostream define 2 member functions get() and put() respectively to handle the single character input/output operations.
- There are 2 types of get() functions.

i, get(char *) : used to read the single character from the input device.

ii, get(void) : used to fetch a character including the blank space, tab and the newline character.

Syntax: get(char), calling function is cin.get(c);

Example:

```
char c;
cin.get(c);
while (c!='\n')
{
    cout<<c;
    cin.get(c);
}
```

(i) put() : This function is used to write a single character to the output device.

- It is a member of ostream class.

Syntax : put(char);

(i) cout.put('x');
displays the character x and

(ii) cout.put(ch); - displays the value of variable ch

(iii) cout.put(65); → display the number i.e 65

Ex:
```
char c;
cin.get(c);
while(c!='\n')
{ cout.put(c);
  cin.get(c);
}
```

Example program for get() & put() :

```
#include<iostream>
using namespace std;
int main()
{
  int count=0;
  char c;
  cout<<" Input text\n";
  cin.get(c);
  while(c!='\n')
  {
    cout.put(c);
    count++;
    cin.get(c);
  }
  cout<<"\n number of chars = "<<count<<"\n";
  return 0;
}
```

output:

Input : object oriented Programming

output:
object oriented programming
number of characters = 27

iii) getline() and write functions:

- we can read and display a line of text more efficiently using the line-oriented input/output functions getline() and write().
- The getline() function reads a whole line of text that ends with a newline character.

Syntax: cin.getline(line, size);

Ex: char name[20];
cin.getline(name, 20); → size of array.
↘ array name

Example C++ Program:

```
#include<iostream>
using namespace std;
int main()
{
  int size=10;
  char city[20];
  cout<<"enter city name:\n";
  cin>>city;
  cout<<"city name:"<<city<<"\n\n";
  cout<<"enter city name again:\n";
  cin.getline(city,size);
  cout<<"city name now:"<<city<<"\n\n";
  cout<<"enter another city name:\n";
  cin.getline(city,size);
  cout<<"new city name:"<<city<<"\n\n";
  return 0;
}
```

ii write():

- It is used to display the line of characters or text.
- Syntax:

```
cout.write(line, size);
```

- The first argument in the right function is 'line'.
- It is used to represent the name of the string to be displayed.
- The second argument 'size' indicates the number of chars to be displayed.

Example c++ program of write():

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
  char *string1="c++";
  char *string2="programming";
  int m=strlen(string1);
  int n=strlen(string2);
  for(int i=1; i<n; i++)
  {
    cout.write(string2, i);
    cout<<"\n";
  }
  for(i=n; i>0; i--)
  {
    cout.write(string2, i);
    cout<<"\n";
  }
  cout.write(string1, m).write(string2, n);
  cout<<"\n";
  cout.write(string1, 10);
  return 0;
}
```

output:

```
p
pr
pro
prog
progr
progra
program
programm
programmi
programmin
programming
programmin
programmi
programm
program
progra
progr
prog
pro
pr
p
c++ programming
c++ prog
```

⇒ Formatted console I/O operations:

- C++ supports a number of features that could be used for formatting the output. These features include.
  - i, ios class functions and flags
  - ii, Manipulators.
  - iii, user-defined output functions.
- The ios class contains a large number of member functions that would help us to format the output in a number of ways.
- The most important ones among them are listed in table.

Function	Task.
a. width()	To specify the required field size for displaying an output value.
b. Precision()	To specify the number of digits to be displayed after the decimal point of a float value.
c. fill()	To specify a character that is used to fill the unused portion of a field.
d. setf()	To specify format flags that can be control the form of output display (such as left and right justification)
e. unsetf()	To clear the flags specified.

a. width()

Syntax: cout.width(w); w is the width in number of columns.

→ The output will be printed in a field of w characters wide at the right end of the field.

→ The width() can specify the field width for only one item.

Ex:1
```
cout.width(5);
cout << 543 << 12 << "\n";
```

output

		5	4	3	1	2

- The value 543 is printed right-justified in the first 5 columns.

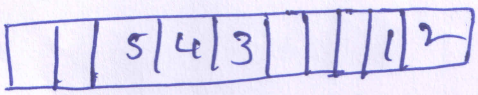
→ The specification width(5) does not retain the setting for printing the number 12. in the example 1

- This can be improved in Example 2 as follows.

```
cout.width(5);
cout << 543;
cout.width(5);
cout << 12 << "\n";
```

→ Example: 2

output:



(b) precision():

- By default, the floating numbers are printed with 6 digits after the decimal point.

Syntax: cout.precision(d);

or 'd' specify the no. of digits to the right of the decimal point.

Ex:1
```
cout.precision(3);
cout << sqrt(2) << "\n";
cout << 3.14159 << "\n";
cout << 2.50032 << "\n";
```

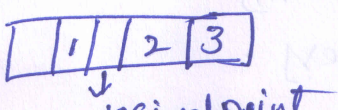
output:

1.141 (truncated)
3.142 (rounded)
2.5 (no trailing zeros)

Ex:2 : we can also combine the field specification with the precision setting.

```
cout.precision(2);
cout.width(5);
cout << 1.2345;
```

output:



5. Filling and padding : fill()

- The unused position of the field are filled with white space, by default.
- However, we can use the fill() to fill the unused position by any desired character.

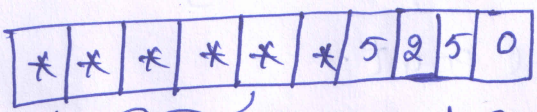
Syntax : cout.fill(ch);

↘ character is used for filling the unused positions.

Example :

```
cout.fill('*');
cout.width(10);
cout << 5250 << "\n";
```

output :



→ unused spaces are filled with '*'.

ii, Manipulators :

- Manipulators are operators used in c++ for formatting output.
- The data is manipulated by the programmers choice of display.
- There are numerous Manipulators available in C++
- Some of the more commonly used manipulators are provided here below.
- Manipulators are functions specifically designed to be used in conjunction with the insertion(<<) and extraction(>>) operators on stream objects.
- Manipulators are used to change formatting parameters on streams and to insert or extract certain special characters.

Manipulators and their Meanings:

Manipulator	Meaning	Equivalent
1. setw(int w)	set the field width to w	width()
2. setprecision(int d)	set the floating point precision to d	precision()
3. setfill(int c)	set the fill character to c	fill()
4. setioflags(long f)	set the format flag f.	setf()
5. resetiosflags(long f)	clear the flag by f	unsetf()
6. endl	Insert new line and flush stream	"\n"

- The header file iomanip provides a set of functions called manipulators which can be used to manipulate the output formats.
- They provide the same features as that of the ios member functions and flags.
- Two or more manipulators can be used as a chain in one statement.

Ex: cout << manip1 << manip2 << manip3 << item;
cout << manip1 << item1 << manip2 << item2;

- This kind of concatenation is useful when we want to display several columns of output.

Some Examples of Manipulators are:

Ex:1 cout << setw(10) << 12345;

- This stmt prints the value 12345 right-justified in a field width of 10 characters.
- The o/p can be made left-justified by modifing the statement as follows.

Ex:2 cout << setw(10) << setiosflags(ios::left) << 12345;

- one stmt can be used to format o/p for 2 or more values.

Ex: 3

```
cout << setw(5) << setprecision(2) << 1.2345
     << setw(10) << setprecision(4) << sqrt(2)
     << setw(15) << setiosflags(ios::scientific) << sqrt(3).
     << endl;
```

- It will print the three values in one line with the field sizes of 5, 10, and 15 respectively.

- We can jointly use the manipulators and the ios functions in program the valid code segment is

```
cout.setf(ios::showpoint);
cout.setf(ios::showpos);
cout << setprecision(4);
cout << setiosflags(ios::scientific);
cout << setw(10) << 123.45678;
```

iii, Designing our own Manipulators (user defined Manipulators)

- we can design our own manipulators for certain special purposes. The general form for creating a manipulator with out any arguments is

Syntax:

```
ostream & manipulator (ostream & output)
{
    .....
    ..... } code
    .....
    return output;
}
```

Ex:

```
ostream & unit (ostream & output)
{                    manipulator name
  output << " inches";
  return output;
}

cout << 36 << unit;
```

- It will produce the following output.

  36 inches.

⇒ Formating Flags, Bit fields and setf():

- Bit fields are the memory space to the particular flag variable.
- setf(), a member function of the ios class, can provide usual practice to Print the text left-justified and many other formating questions.
- The setf() function can be used as follows.

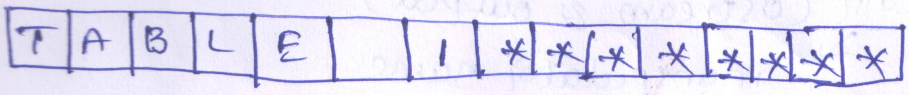
Syntax: cout.setf(arg1, arg2)

- The arg_1 is one of the formating flags defined in the class ios.
- The formating flag specifies the format action required for the output.
- The arg_2 as a bitfield specifies the group to which the formatting flag belongs.

Ex:

```
cout.fill('*');
Cout.setf(ios::left, ios::adjustfield);
cout.width(15);
Cout<< "TABLE 1" << "\n";
```

• This will produce the following output



The statements:

```
cout.fill('*');
cout.precision(3);
cout.setf(ios::internal, ios::adjustfield);
cout.setf(ios::scientific, ios::floatfield);
cout.width(15);
cout<<-12.34567<<"\n";
```

→ will produce the following output:

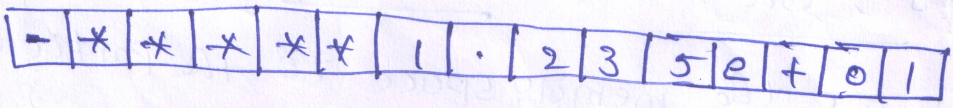


Table for Flags and bit fields for setf() function:

Format required	Flag (arg1)	Bit-field (arg2)
Left-justified output	ios::left	ios::adjustfield
Right-justified output	ios::right	ios::adjustfield
Padding after sign or base Indicator (like +## 20)	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal base	ios::dec	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal base	ios::hex	ios::basefield

• That the first argument should be one of the group members of the second argument.

⇒ Flags without Bit Field: (01) Displaying Trailing zero's and play Sign:

- The setf() can be used with the flag ios::showPoi as a single argument to achive this form of output.

Ex: cout.setf(ios::showpoint); // display trailing zeros.

Similarly, a play sign can be printed before a positive num using the following statement.

cout.setf(ios::showpos); // show + sign

for Example.

```
cout.setf(ios::showpoint);
cout.setf(ios::showpos);
cout.precision(3);
cout.setf(ios::fixed, ios::floatfield);
cout.setf(ios::internal, ios::adjustfield);
cout.width(10);
cout << 275.5 << "\n";
```

output:

+			2	7	5	.	5	0	0

- The flags such as showpoint and showpos do not have any bit fields and therefore are used as single argument in setf().

- The following table shows list the flags that do not possess a named bit field.

Flag	Meaning
ios::showbase	use base indicator on output
ios::showpos	Print + before positive numbers
ios::showpoint	show trailing decimal point and zeros
ios::uppercase	use uppercase letters for hex output
ios::skipws	skip white space on input.
ios::unitbuf	Flush all streams after insertion
ios::stdio	Flush stdout and stderr after insertion.