Object Oriented Programming

Prof. Zankhana Dabhi
Department of Information Technology,
D. D. University, Nadiad.

What is Procedure Oriented Programming (POP)?

- POP follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions.
- The program is divided into small parts called functions and then it follows a series of computational steps to be carried out in order.
- It follows a top-down approach to actually solve a problem. Dividing the program into functions is the key to procedural programming.

Strengh of C

- Building block for many other programming languages.
- Powerful and efficient language
- Portable language
- Built-in functions
- Quality to extend itself
- Structured programming language

Limitations of C

- C provides no data protection.
- C doesn't support Object Oriented Programming(OOP) features like Inheritance, Encapsulation, Polymorphism etc.
- C does not implement the concept of namespaces.
- C does not have any constructor or destructor.

Concept of namespace

```
int main()
{
  int value;
  value=0;
  double value;
  value=0.0;
}
```

Contd..

```
namespace first
{
   int value=50;
}
int value=10;
int main()
{
   int value=20;
   cout<<first::value;
}
```

What is Object Oriented Programming (OOP)?

- OOP is a high-level programming language where a program is divided into small chunks called objects using the object-oriented model, hence the name. This paradigm is based on objects and classes.
- Object – An object is basically a self-contained entity that accumulates both data and procedures to manipulate the data. Objects are merely instances of classes.
- Class – A class, in simple terms, is a blueprint of an object which defines all the common properties of one or more objects that are associated with it. A class can be used to define multiple objects within a program.
- The OOP paradigm mainly eyes on the data rather than the algorithm.
- The modules cannot be modified when a new object is added.
- Objects can communicate with each other through same member functions. This process is known as message passing.

Difference between OOP and POP

- Definition
  - OOP stands for Object-oriented programming and is a programming approach that focuses on data rather than the algorithm, whereas POP, short for Procedure-oriented programming, focuses on procedural abstractions.
- Programs
  - In OOP, the program is divided into small chunks called objects which are instances of classes, whereas in POP, the main program is divided into small parts based on the functions.
- Focus
  - The main focus is on the data associated with the program in case of OOP while POP relies on functions or algorithms of the program.
- Accessing Mode
  - Three accessing modes are used in OOP to access attributes or functions – ‘Private’, ‘Public’, and ‘Protected’. In POP, on the other hand, no such accessing mode is required to access attributes or functions of a particular program.

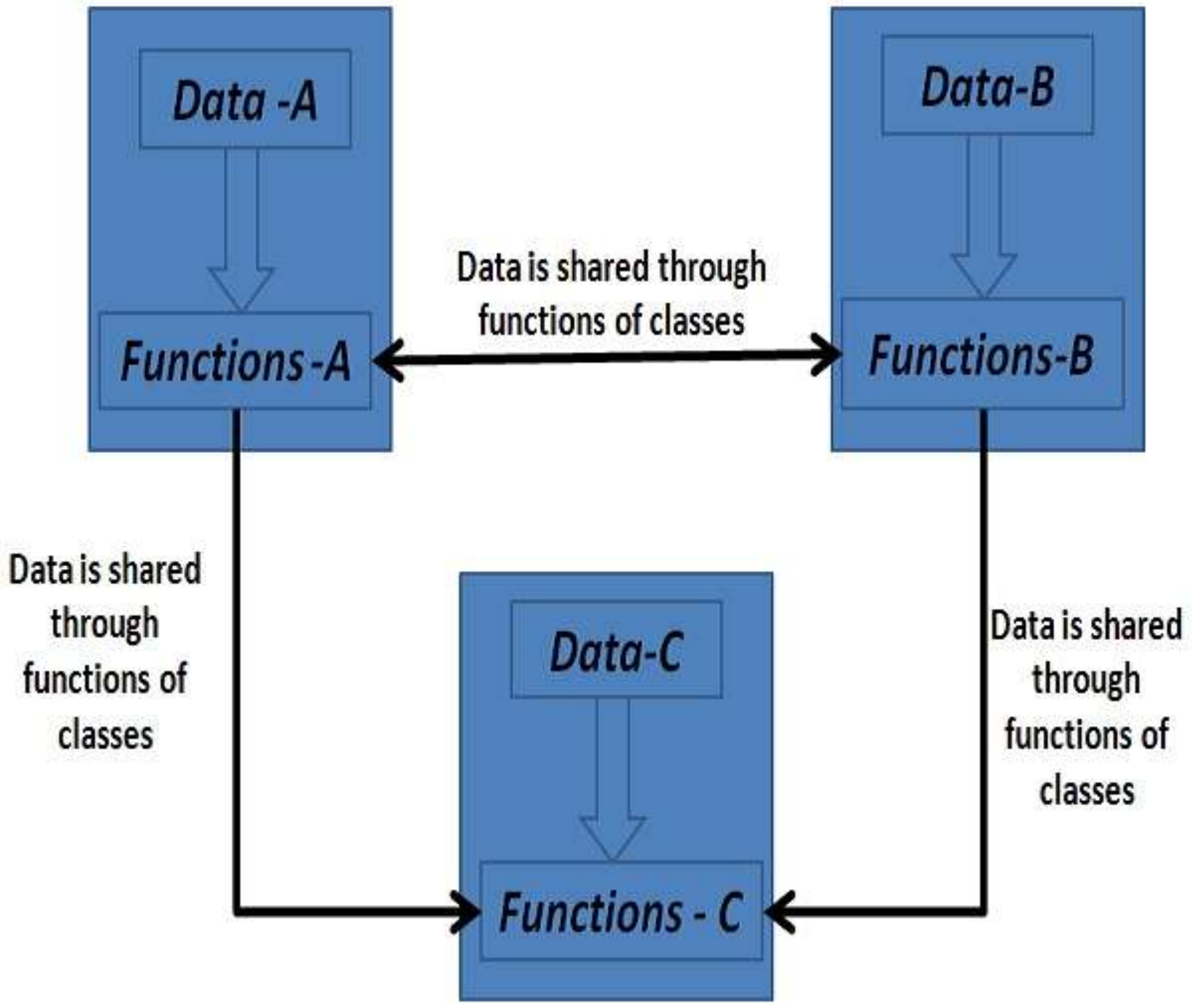
Difference between OOP and POP

- Execution
  - In OOP, various functions can work simultaneously while POP follows a systematic step-by-step approach to execute methods and functions.
- Data Control
  - In OOP, the data and functions of an object act like a single entity so accessibility is limited to the member functions of the same class. In POP, on the other hand, data can move freely because each function contains different data.
- Security
  - OOP is more secure than POP, thanks to the data hiding feature which limits the access of data to the member function of the same class, while there is no such way of data hiding in POP, thus making it less secure.

Contd.

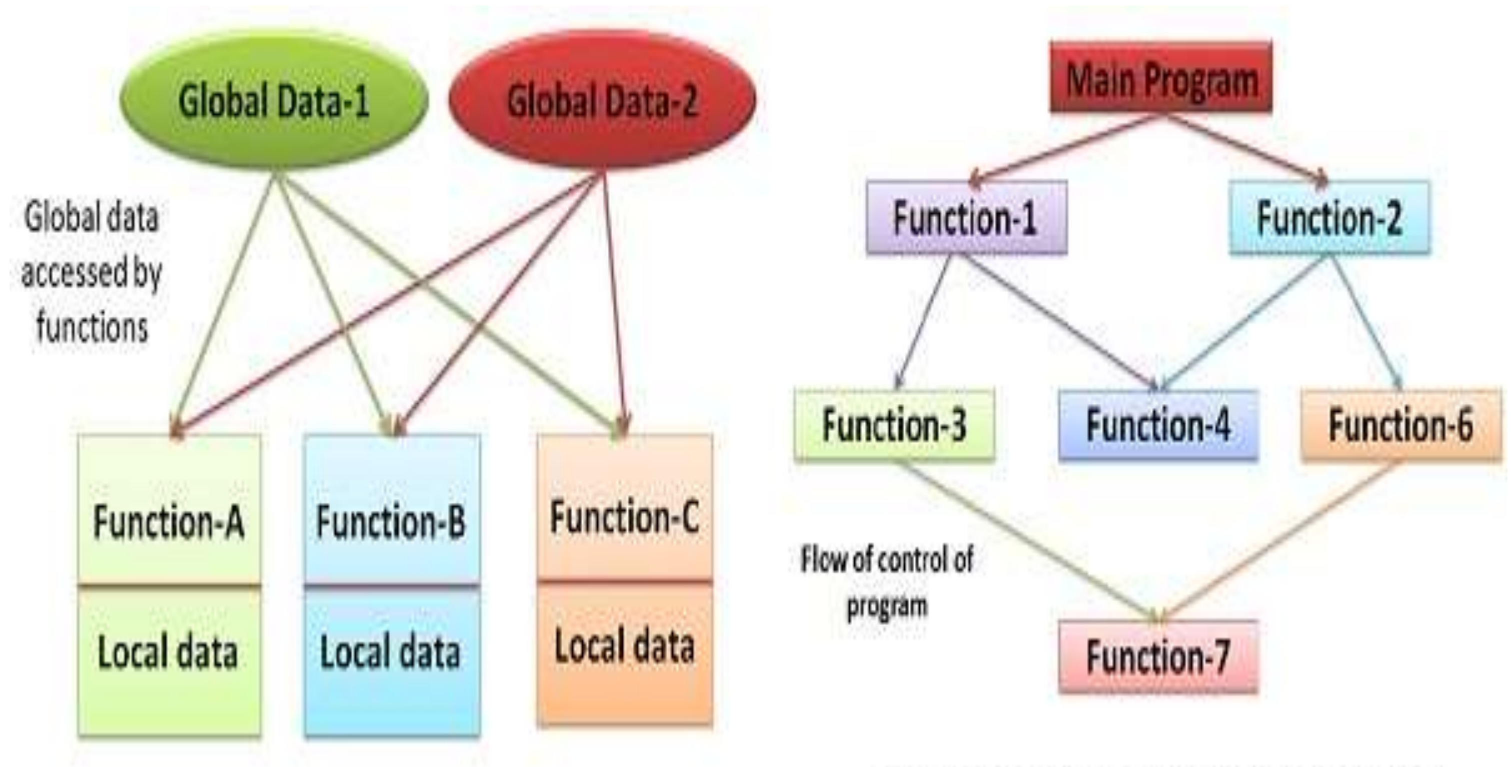
- Ease of Modification
  - New data objects can be created easily from existing objects making object-oriented programs easy to modify, while there's no simple process to add data in POP, at least not without revising the whole program.
- Process
  - OOP follows a bottom-up approach for designing a program, while POP takes a top-down approach to design a program.

Data Flow in Object-Oriented Programming


Data -A
Functions -A
Data-B
Functions-B
Data is shared through functions of classes
Data is shared through functions of classes
Data-C
Functions - C
Data is shared through functions of classes

Data flow in object-oriented programming

Structure of Procedure Oriented Program

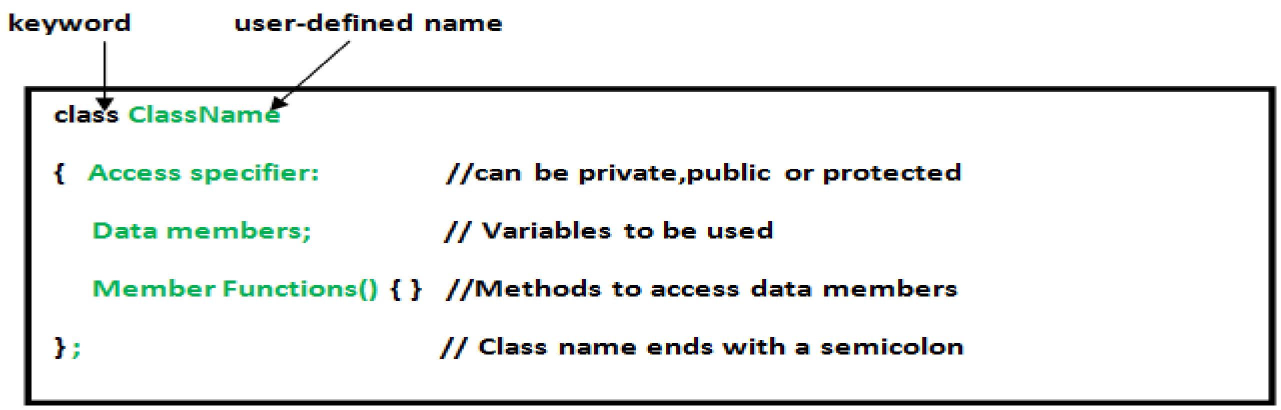


Structure of procedure oriented program

OOP Characteristics

- **Classes**
  - It is a set of objects of similar type. A complete set of data and code of an object creates an user-defined data type by using a class.
  - A Class is a user defined data-type which has data members and member functions.
  - For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc.
- **Objects**
  - It is considered as a variable of type class and an instance of a class.

Defining class and declaring objects



```
class ClassName
{  Access specifier:          //can be private,public or protected
   Data members;              // Variables to be used
   Member Functions() { }     //Methods to access data members
};                            // Class name ends with a semicolon
```

- Declaring Objects:
  - When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.
- Syntax:

  ClassName ObjectName;
- Accessing data members and member functions:
  - The data members and member functions of class can be accessed using the dot('.') operator with the object.

OOP Characteristics

- **Data abstraction** :
  - Abstraction is nothing but a method of hiding background details and representing essential features.
  - Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.
  - Abstraction using Classes**:**
    - Class helps us to group data members and member functions using available access specifiers.
    1. Members declared as **public** in a class, can be accessed from anywhere in the program.
    2. Members declared as **private** in a class, can be accessed only from within the class.
  - Abstraction in Header files**:**
    - One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file.

Contd…

```
class implementAbstraction
{
    private:
        int a, b;

    public:
        void set(int x, int y)            // method to set values of  private members
        {
            a = x;
            b = y;
        }

        void display()
        {
            cout<<"a = " <<a << endl;
            cout<<"b = " << b << endl;
        }
};
 int main()
{
    implementAbstraction obj;
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

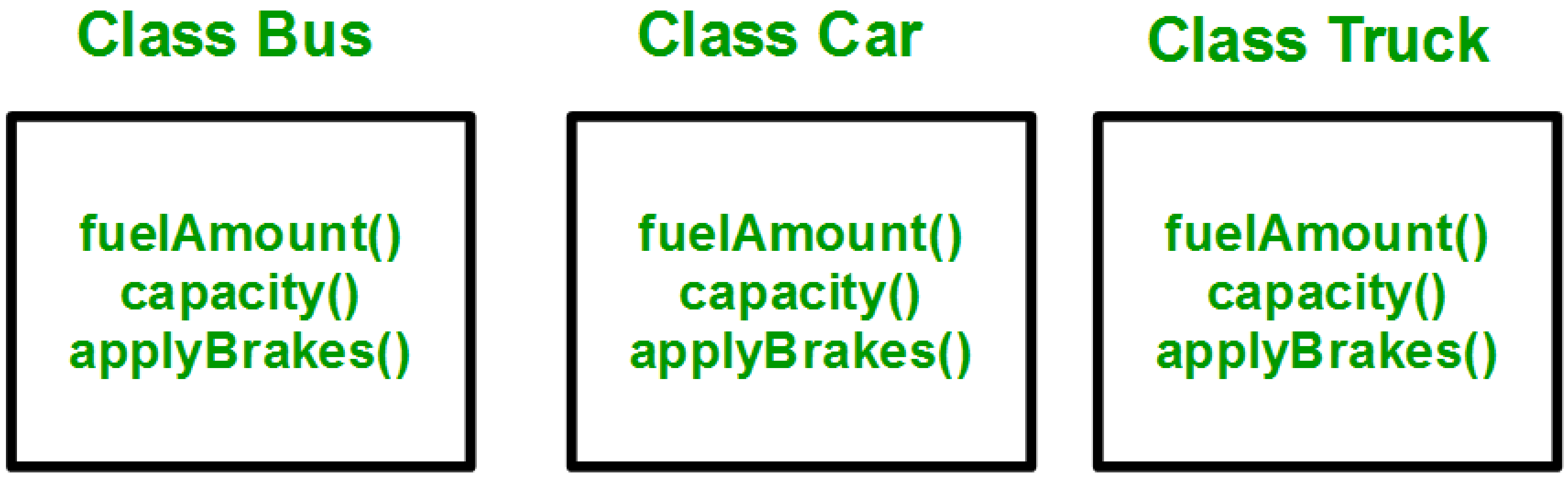
Contd…

- **Data Encapsulation**:
  - All C++ programs are composed of two fundamental elements − data and code.
- Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.
- Data encapsulation led to the important OOP concept of **data hiding**.

Contd..

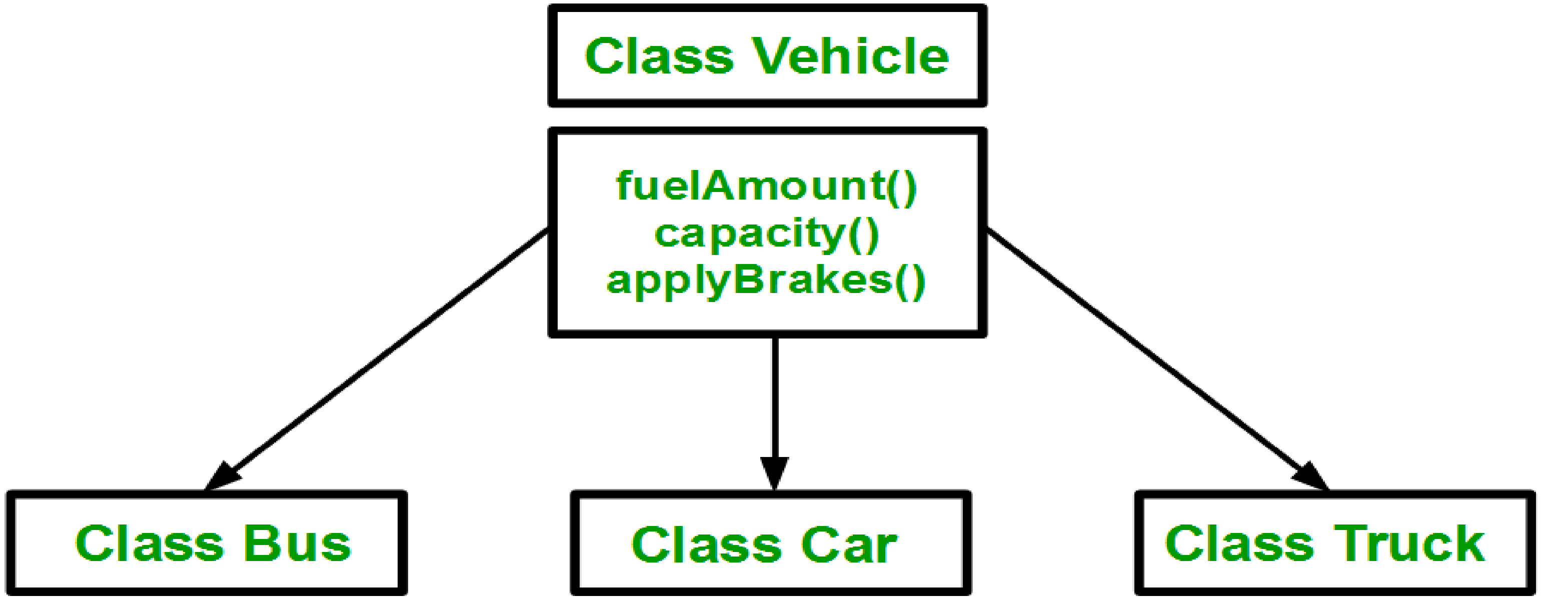
- Inheritance:
  - The capability of a class to derive properties and characteristics from another class is called **Inheritance**
  - Sub Class: The class that inherits properties from another class is called Sub class or Derived Class. Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

Inheritance



- The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes.
- This process results in duplication of same code 3 times. This increases the chances of error and data redundancy.

Inheritance

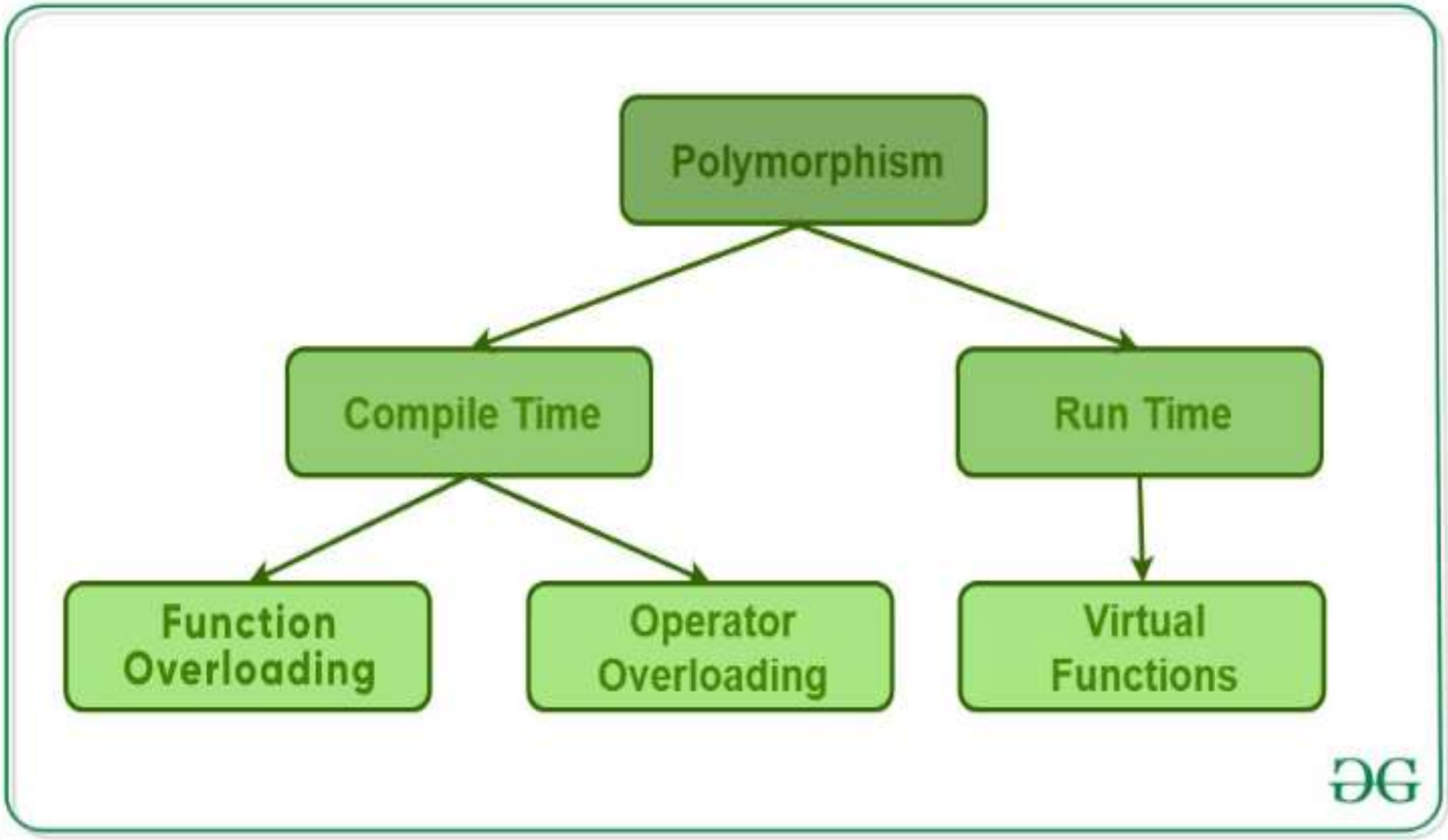


Using inheritance, we have to write the functions only one time instead of three times as we have inherited rest of the three classes from base class(Vehicle).

OOP Charactreristics

- **Polymorphism:**
  - It is combination of two words- poly and morphism
  - polymorphism means having many forms.
- we can define polymorphism as the ability of a message to be displayed in more than one form.
- **In C++ polymorphism is mainly divided into two types:**
- Compile time Polymorphism
- Runtime Polymorphism

Contd…


Polymorphism
Compile Time
Run Time
Function Overloading
Operator Overloading
Virtual Functions

Compile Time Polymorphism

- Function Overloading:
  - When there are multiple functions with same name but different parameters then these functions are said to be **overloaded**.
  - Functions can be overloaded by **change in number of arguments** or/and **change in type of arguments**.

Contd…

```
class Geeks
{
   public:

   void func(int x)              // function with 1 int parameter
   {
      cout << "value of x is " << x << endl;
   }

   void func(double x)       // function with same name but 1 double parameter
   {
      cout << "value of x is " << x << endl;
   }

   void func(int x, int y)          // function with same name and 2 int parameters
   {
      cout << "value of x and y is " << x << ", " << y << endl;
   }
};
```

•

Contd…

```
int main() {

    Geeks obj1;

    // Which function is called will depend on the parameters passed
    // The first 'func' is called

    obj1.func(7);

    // The second 'func' is called

    obj1.func(9.132);

    // The third 'func' is called

    obj1.func(85,64);
    return 0;
}
```

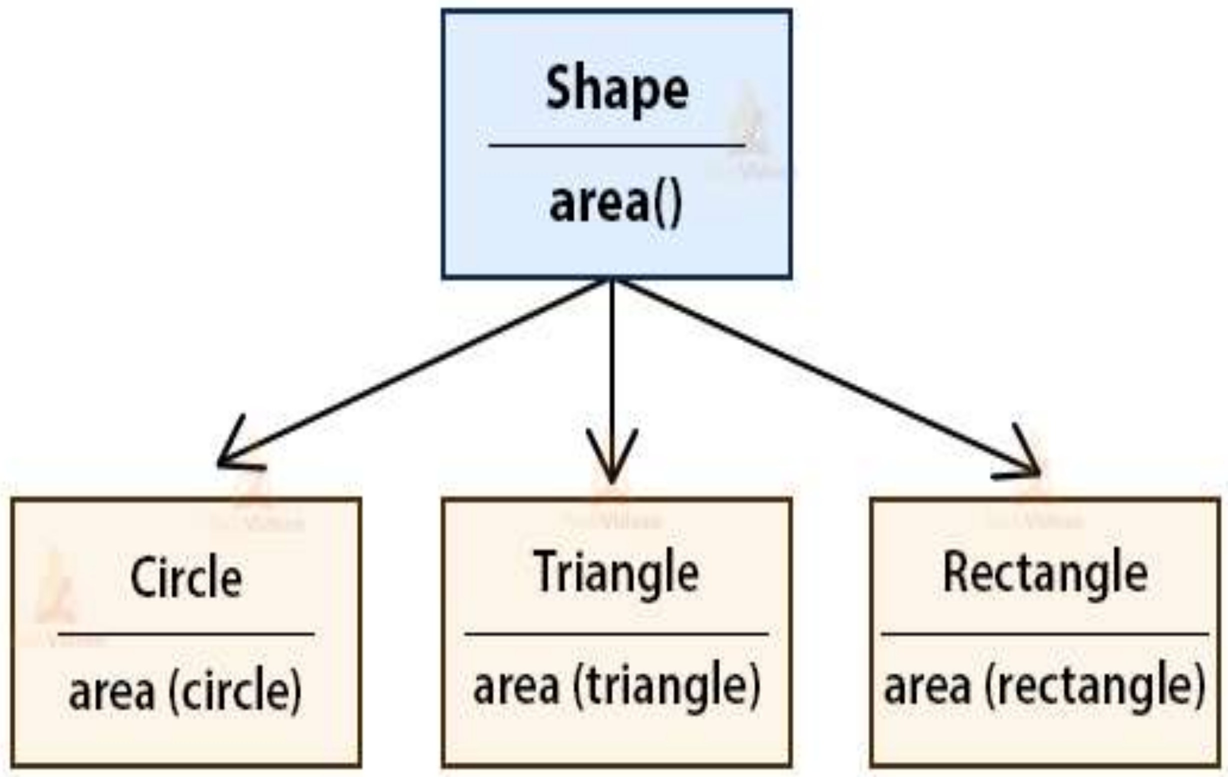
Contd…

- **Operator Overloading:**
  - C++ also provide option to overload operators.
  - we can make the operator (‘+’) for string class to concatenate two strings.
  - single operator ‘+’ when placed between integer operands , adds them and when placed between string operands, concatenates them.
  - **Example:**
  - Complex c3 = c1 + c2;   // An example call to "operator+"

Run Time Polymorphism

- It is the ability of objects of different types to respond to functions of the same name.
- The user does not have to know the exact type of the object in advance.
- The behavior of the object can be implemented at runtime.
- Function Overriding:
  - occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

Run Time Polymorphism



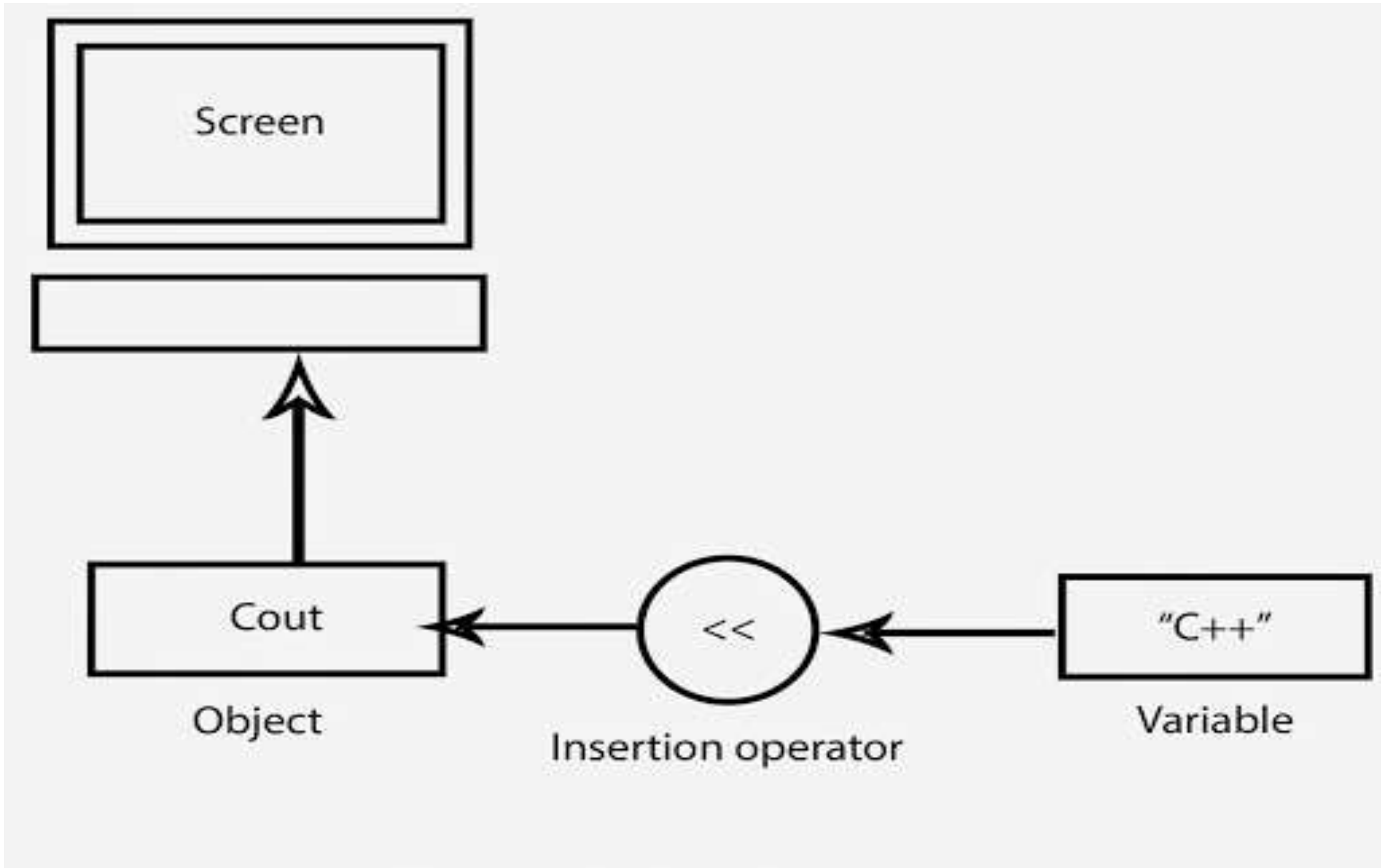
Your First C++ Program

```
# include<iostream>
using namespace std;
int main()
{
   cout<<“Hello World”;
}
```

- C++ compiler ignores white space (space, carriage returns, linefeed, tabs)

Output using cout

- The identifier cout is actually an object which is predefined to correspond to the standard output stream.
- The << operator is called insertion or put to operator.
- << operator directs the contents of the variable on its right to the object on its left.



Output using insertion operator

Preprocessor Directives

- # include <iostream>
- Program statements are instructions to the computer, on the other hand, preprocessor directive is an instruction to the compiler itself.
- A part of the compiler is called preprocessor.
- #include directive
  - It tells the compiler to insert one file to the source file.

Comments

- Comment Syntax:

```
# include<iostream>  // preprocessor directive
using namespace std;
void main()
{                        // start function body
   cout<<“Hello World”;
}                        // end function body
```

Contd..

- Second Comment Style in c++:

```
/*  this is an old style comment  */
```

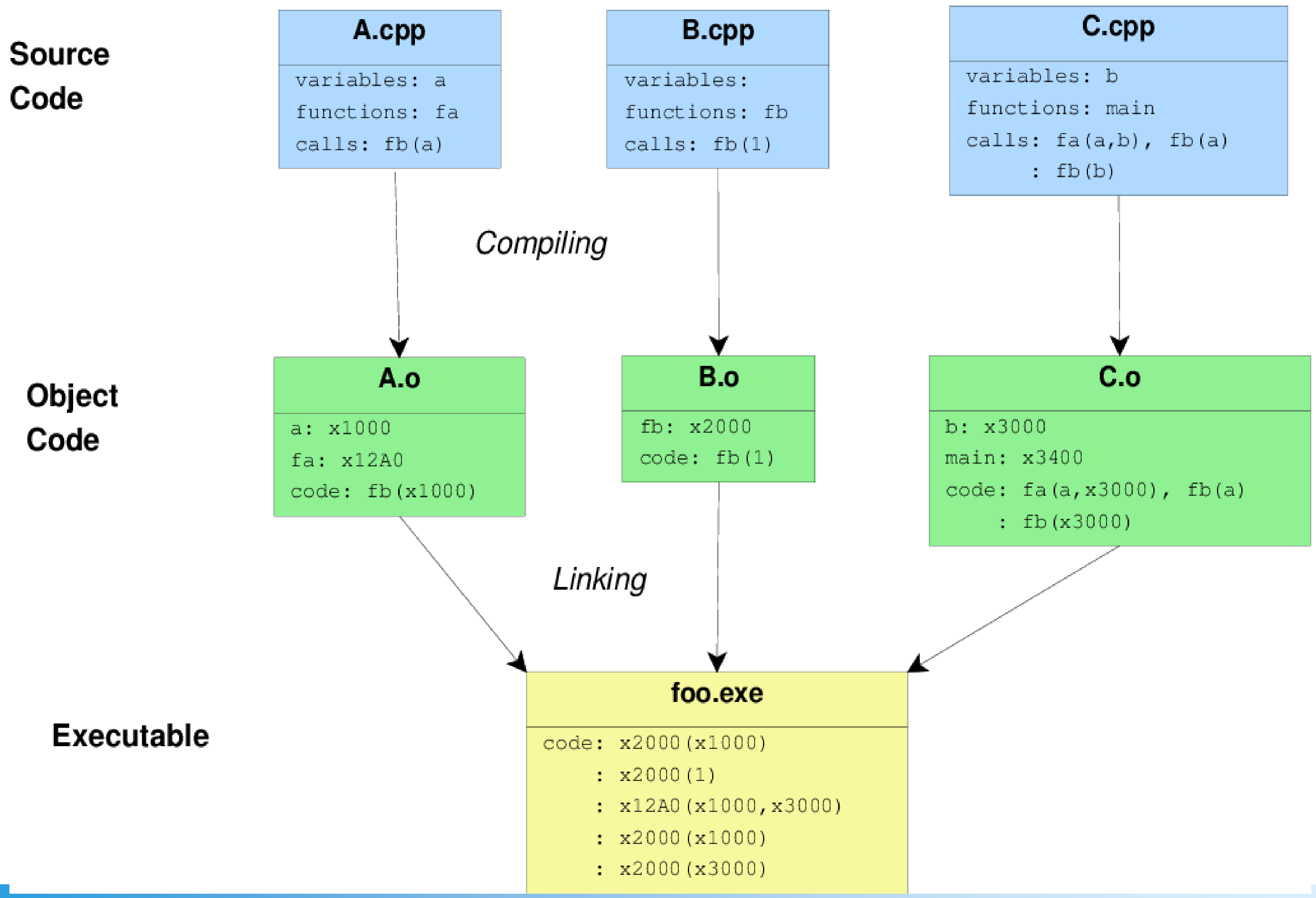
- Multiline comment:

```
/* this
   is a
   potentially very
   long multiline
   comment
 */
```

Saving Your Program

- C++ program has .cpp file extension.
- Source file is not an executable program.
- Transforming source file into executable file requires two steps: compiling and linking.
- After compilation the source file is converted to object file which contains machine-language instructions.
- The linking step is necessary because an executable program may consist of more than one object file.
- Linking combines the object files into a single executable program.

Contd..



Source Code
Object Code
Executable
Compiling
Linking
A.cpp
variables: a
functions: fa
calls: fb(a)
B.cpp
variables:
functions: fb
calls: fb(1)
C.cpp
variables: b
functions: main
calls: fa(a,b), fb(a)
: fb(b)
A.o
a: x1000
fa: x12A0
code: fb(x1000)
B.o
fb: x2000
code: fb(1)
C.o
b: x3000
main: x3400
code: fa(a,x3000), fb(a)
: fb(x3000)
foo.exe
code: x2000(x1000)
: x2000(1)
: x12A0(x1000,x3000)
: x2000(x1000)
: x2000(x3000)

Input with cin

- cin is an object of istream class which corresponds to the standard input stream.
- This stream represents data coming from keyboard.
- The >> is called extraction or get from operator.
- It takes the value from the stream object on its left and places it in the variable on its right.

Example

```
#include <iostream>
using namespace std;
int main()
{
        int x, y, z;

        /* For single input */
        cout << "Enter a number: ";
        cin >> x;

        /* For multiple inputs*/
        cout << "Enter 2 numbers: ";
        cin >> y >> z;

        cout << "Sum = " << (x+y+z);
        return 0;
 }
```

Example

```
#include <iostream>

int main()
{
         int x, y, z;

        /* For single input */
        std::cout << "Enter a number: ";
        std::cin >> x;

        /* For multiple inputs*/
        std::cout << "Enter 2 numbers: ";
        std::cin >> y >> z;

        std::cout << "Sum = " << (x+y+z);
        return 0;
 }
```

Const Qualifier

- const qualifier is used to declare a variable as constant.
- Example:

```
# include <iostream>
int main()
{
 const int x = 10;
 x = 12;                 // error
 return 0;
 }
```

- Pointer to constant:

  const int *ptr;   **OR**

  int const *ptr;

- We can change the pointer to point to any other integer variable, but cannot change the value of the object (entity) pointed using pointer ptr.

Pointer to constant

```
#include <iostream>
int main()
{
    int i = 10;
    int j = 20;

    const int *ptr = &i;

    cout<< *ptr;

    *ptr = 100;                  //error

    ptr = &j;                   /* valid */

    cout<< *ptr;

    return 0;
}
```

- Constant pointer to variable:

```
int *const ptr;
```

- We can change the value of object pointed by pointer, but cannot change the pointer to point another variable.

```
#include <iostream>

int main(void)
{
        int i = 10;
        int j = 20;

        int *const ptr = &i;

        cout<< *ptr;

        *ptr = 100;    /* valid */

        cout<< *ptr;

        ptr = &j;        /* error */
        return 0;
}
```

- constant pointer to constant:

```
const int *const ptr;
```

- we cannot change value pointed by the pointer as well as we cannot point the pointer to other variables.

```
#include <iostream>

int main(void)
{
    int i = 10;
    int j = 20;

    const int *const ptr = &i;

    cout<< *ptr;

    ptr = &j;          /* error */
    *ptr = 100;        /* error */

    return 0;
}
```

Manipulators

- Manipulators are operators used in C++ for formatting output.
- endl manipulator
- setw manipulator
- setfill manipulator

Manipulators

- endl manipulator:
  - This manipulator has the same functionality as the ‘n’ newline character.
- *For example:*
- 1. cout << "Exforsys" << endl;
  2. cout << "Training";

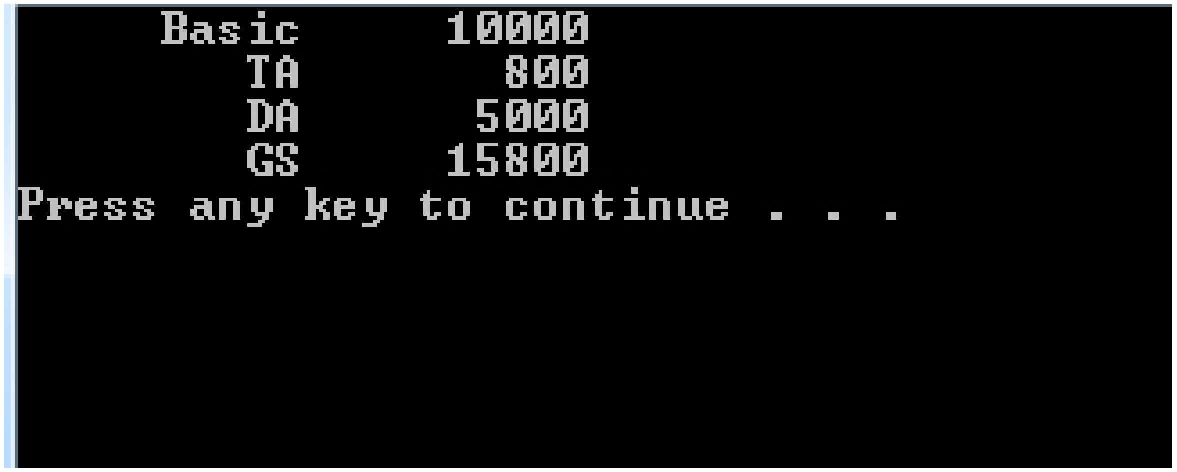
Manipulators

- setw manipulator:
  - This manipulator sets the minimum field width on output.
  - setw causes the number or string that follows it to be printed within a field of x characters wide and x is the argument set in setw manipulator.

#include <iomanip>

Manipulators

```
#include <iostream>
#include <iomanip>
void main( )
{
    int x1=10000,x2= 800, x3=5000,x4=15800;
   cout << setw(8) << “Basic" << setw(20) << x1 << endl
         << setw(8) << “TA"     << setw(20) << x2 << endl
         << setw(8) << “DA “     << setw(20) << x3 << endl
         << setw(8) << “GS"     << setw(20) << x4 << endl;
}
```



```
   Basic               10000
      TA                 800
      DA                5000
      GS               15800
Press any key to continue . . .
```

Manipulators

- setfill manipulator :

- This is used after setw manipulator.
- If a value does not entirely fill a field, then the character specified in the setfill argument of the manipulator is used for filling the fields.

Manipulators

- *Example:*

```
#include <iostream>
#include <iomanip>
int main()
{
   cout << setw(15) << setfill('*') << 99 << 97 << endl;
   return 0;
}
```

Output: ********9997

Type Conversion

- A type cast is basically a conversion from one type to another.
- There are two types of type conversion:
- Implicit Type Conversion
- Explicit Type Conversion

Type Conversion

- Implicit Type Conversion:
  - Also known as 'automatic type conversion'
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.
  - bool -> char -> int -> long -> long long -> float -> double -> long double

Type Conversion

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

int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    cout << "x = " << x << endl
         << "y = " << y << endl
         << "z = " << z << endl;
  return 0;
}
```

Type Conversion

- Explicit Type Conversion:
  - This process is also called type casting and it is user-defined.
  - Here the user can typecast the result to make it of a particular data type.
  - Syntax:

    (type) expression   OR   type (expression)

Type Conversion

- Example:
- {

```
int a=25000;
a=(a*10)/10;                // result too large:range:-32,768 to 32,767
cout<<“a=“<<a<<endl;            //wrong answer
}
{
 int a=25000;
 a=(long(a)*10)/10;          //cast to long
 cout<<“a=“<<a<<endl;
}
```

Arithmetic Operators

- These are the operators used to perform arithmetic/mathematical operations on operands.
- Examples: (+, -, *, /, %,++,–)
- Arithmetic operator are of two types:
- **Unary Operators**: Operators that operates or works with a single operand are unary operators.
- For example: (++ , –)
- **Binary Operators**: Operators that operates or works with two operands are binary operators.
- For example: (+ , – , * , /)

Contd.

```
int main()     {
    int a = 10, b = 4, res;

    res = a + b;                    // addition
    cout<< res;

    res = a - b;                       // subtraction
    cout<< res;

    res = a * b;                    // multiplication
    cout<< res;

    res = a / b;                    // division
    cout<< res;

    res = a % b;                    // modulus
    cout<< res;

    return 0;
}
```

Arithmetic Operators

- Increment and decrement operator:
  - Pre increment and post increment
    - ++x   and   x++
  - Pre decrement and post decrement
    - --x   and   x--

Arithmetic Operators

-
  ```
  #include<iostream>
  int main()
  {
   int a=10;
   cout<<a;
   cout<<++a;
   cout<<a;
   cout<<a++;
   cout<<a;
  }
  ```

Arithmetic Operators

- Output:

```
10
11
11
11
12
```