Object-Oriented Programming with C++

Introduction to C++

- C++ is an extension of C . In other words C++ is a superset of C
- C++ was invented by Bjarne stroustrup in 1979 at Bell laboratories, New Jersey
- Initially it was called “C with classes”. However in 1983 the name was changed to C++
- The invention of C++ was necessitated by one major programming factor :: *increasing complexity*.
- In C , once a program exceeds from 25000 to 100000 lines of code, it becomes complex & difficult to grasp
- In structured programming language, data types are processed in many functions, and when changes occur in data types, modification must be made at every location that acts on these data types within the program.
- This is a complex task for large sized programs

Very brief history of C++

1962
Simula I was invented by Kristen Nygaard and Ole-Johan Dahl as a *simulation language*

1967
Simula 67 developed as the first *object-oriented* language

1969-1973
The C language was invented by Dennis Ritchie at Bell Labs

1972
D. Ritchie and Ken Thompson re-write the Unix OS in C

1979
Bjarne Stroustrop began developing "C with Classes"

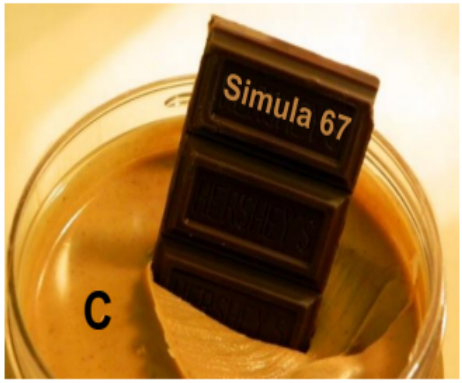
1983
"C with Classes" renamed to C++

1985
1st commercial C++ compiler, Cfront, released by AT&T

1989
C++ 2.0 standard released.

2011
Major update: C++11 standard released

2014
Minor update: C++14 released.



C++

BOSTON UNIVERSITY

For details more check out A History of C++: 1979−1991

Characteristics of C++

- **C++ is…**
  - **Compiled**.
    - A separate program, the compiler, is used to turn C++ source code into a form directly executed by the CPU.
  - **Strongly typed and unsafe**
    - Conversions between variable types must be made by the programmer (strong typing) but can be circumvented when needed (unsafe)
  - **C compatible**
    - call C libraries directly and C code is nearly 100% valid C++ code.
  - **Capable of very high performance**
    - The programmer has a very large amount of control over the program execution
  - **Object oriented**
    - With support for many programming styles (procedural, functional, etc.)
- **No automatic memory management**
  - The programmer is in control of memory usage

- The high-level programming languages are broadly categorized in to two categories:

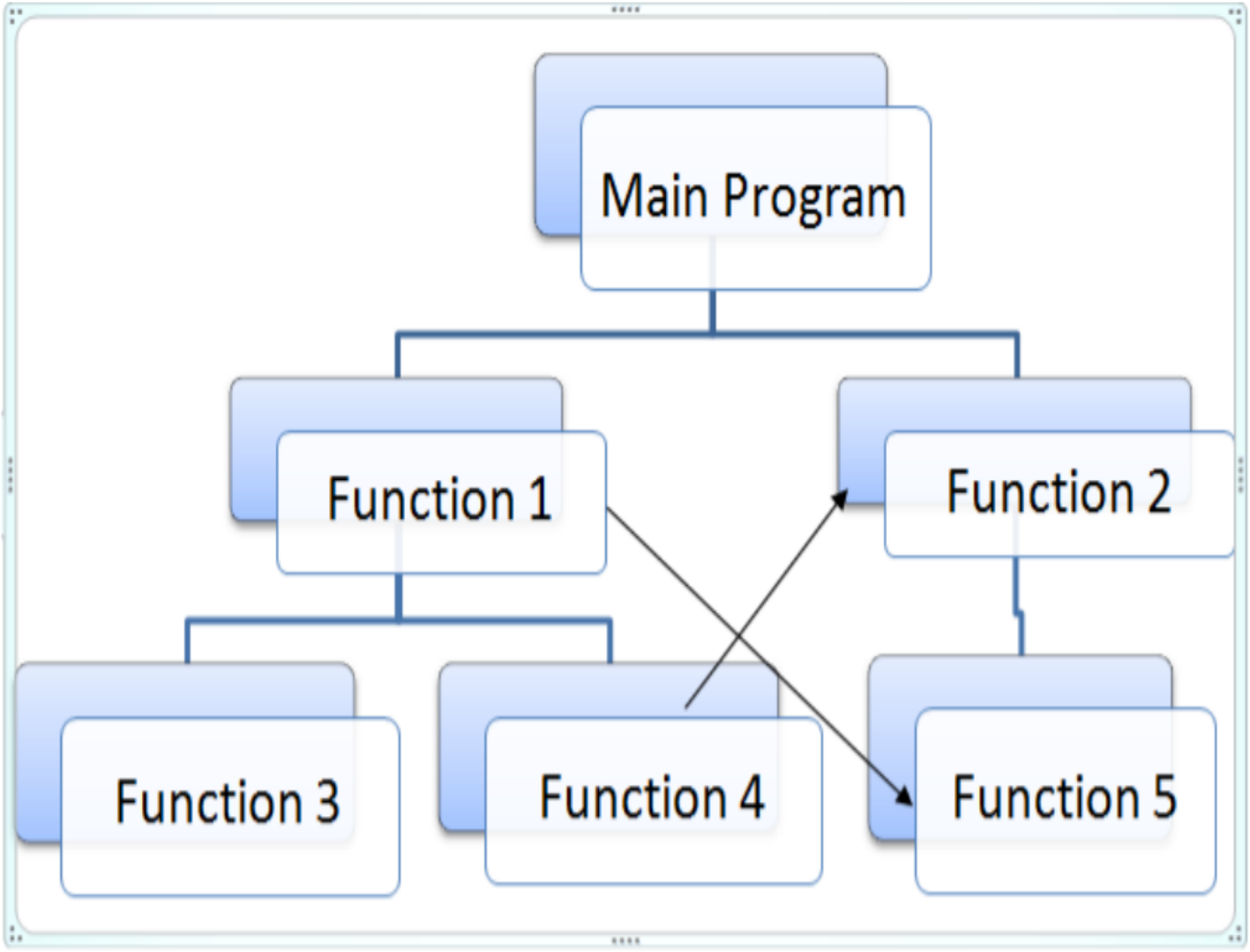
(i) Procedure oriented programming(POP) language.

(ii) Object oriented programming(OOP) language.

What is a procedure-oriented programming?

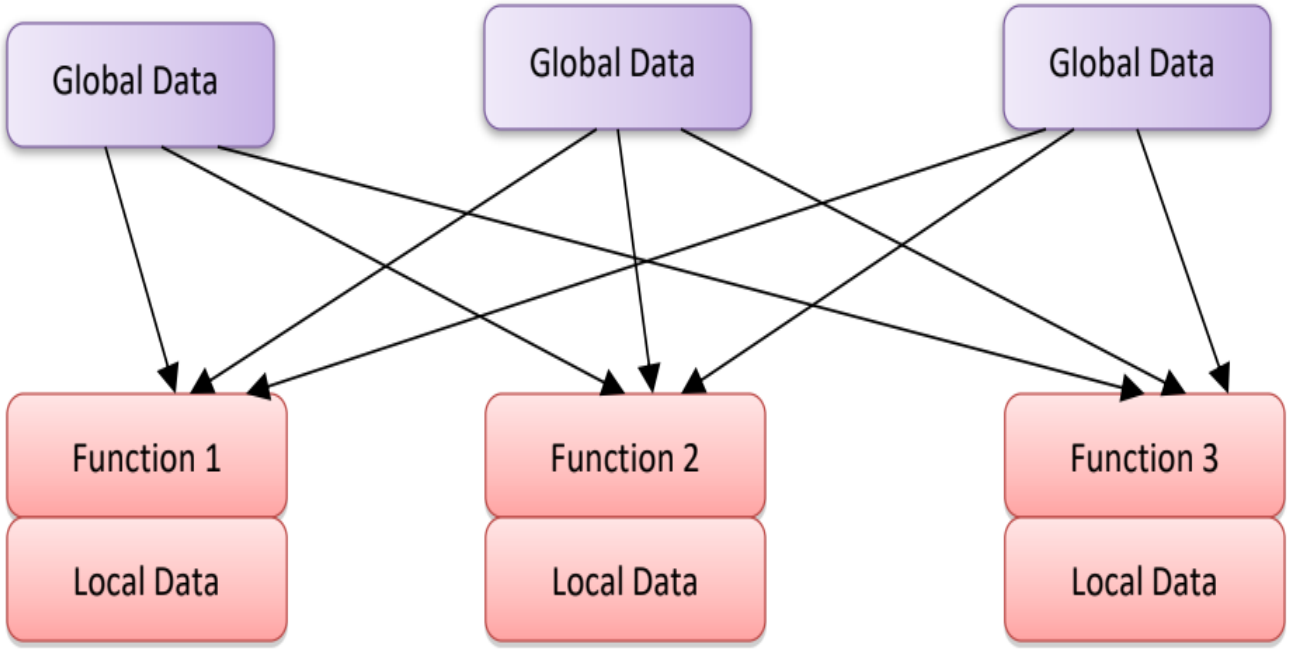
- What is a procedure-oriented programming?
  - Example: COBOL, FORTRAN , C etc.
  - Problem is viewed as a sequence of things to be done
    - Example: Reading, calculating and printing
  - A number of functions are written to accomplish this task
  - The primary focus is on functions
  - Very little attention is given to the data that are being used by functions
    - What happens to the data?
    - How are they affected by the functions that work on them?

Typical Structure Of Procedure Oriented Programs



Procedure-Oriented Programming

- Multi- function program, many important data items are placed as global
  - They may be accessed by all the functions
  - Each function may have its own local data



Relationship of data and functions in procedural programming

Some characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design.

DRAWBACKS:

- In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. ***Global data are more vulnerable to an inadvertent change by a function.***

- In a large program it is very ***difficult to identify what data is to be used by which function***. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides an opportunity for bugs to creep in.

- Another serious drawback is that it ***does not model real world problems*** very well. This is because functions are action oriented and do not really correspond to elements of the problem.

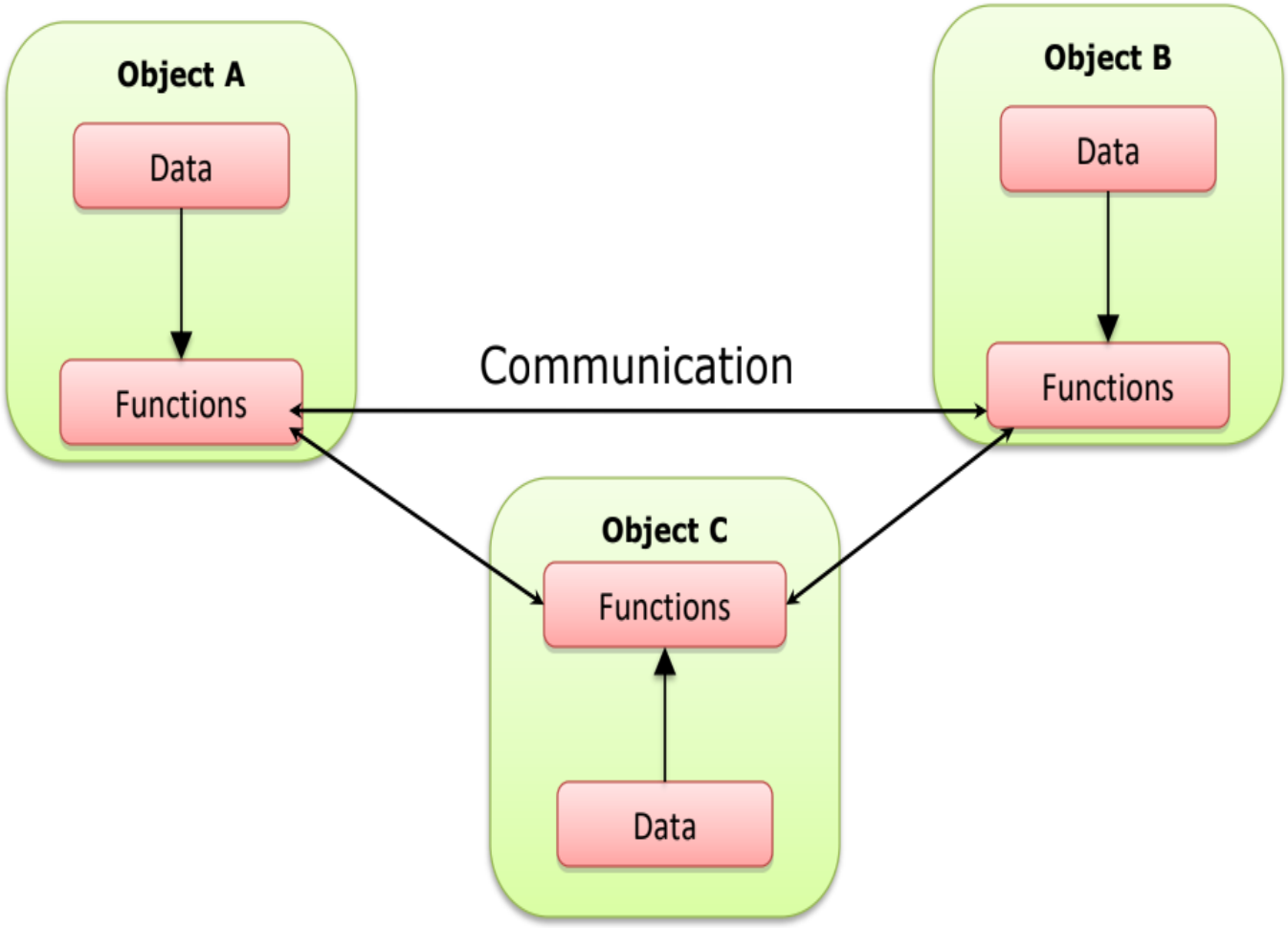
Object-Oriented Programming

Object-oriented programming (OOP) is a way to organize and conceptualize a program as a set of interacting objects.

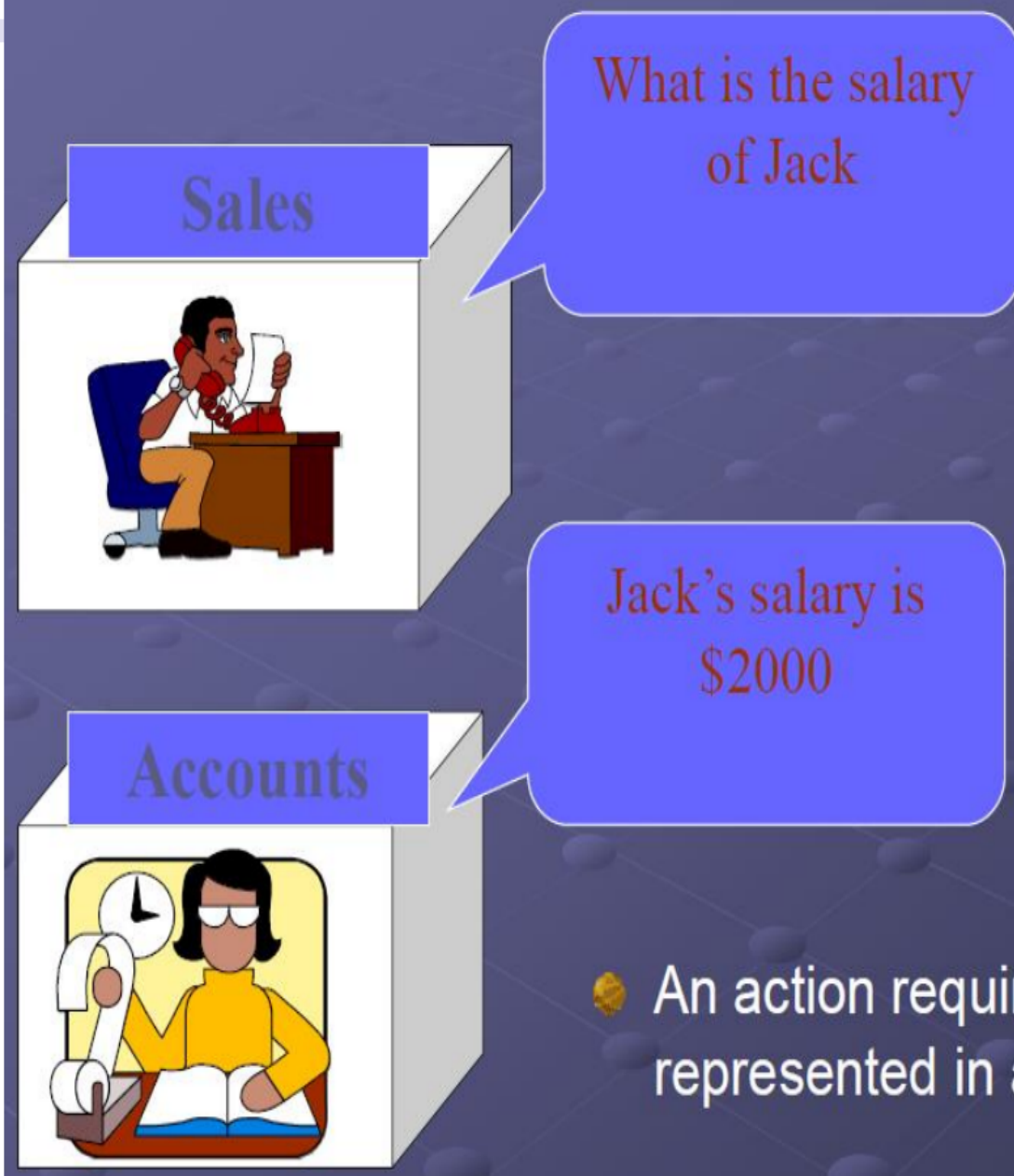
- The programmer defines the types of objects that will exist.
- The programmer creates object instances as they are needed.
- The programmer specifies how these various object will communicate and interact with each other.

Object-Oriented Programming(Cont)

- Data of an object can be accessed only by the functions associated with that object
- Functions of one object can access the functions of other objects



Method



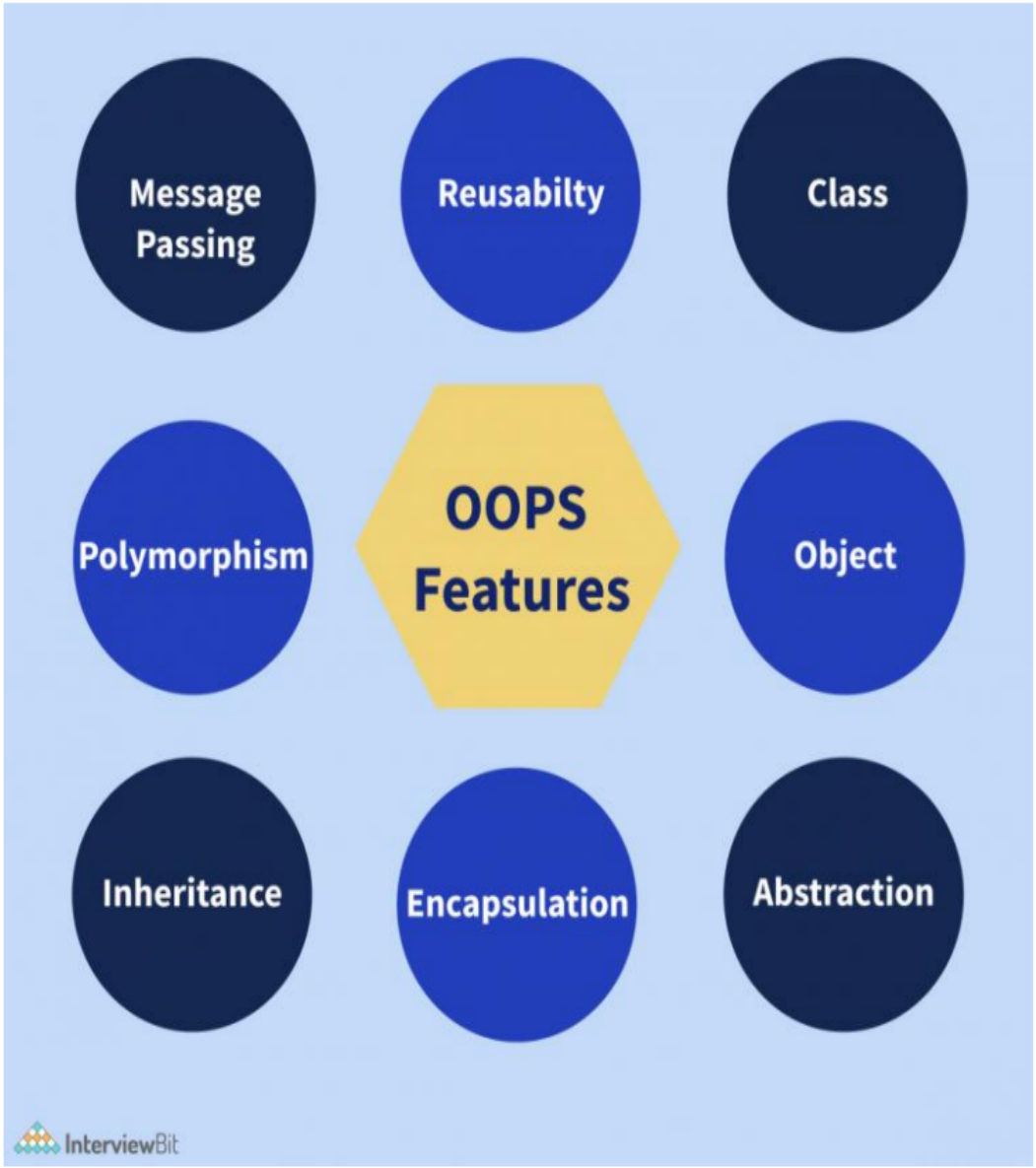
- The black box actually contains code
- The communication between objects is done using messages.
- These messages can be translated to function calls in a program.

- An action required of an object or entity represented in a class is known as a method

Object-Oriented Programming(Cont)

- Striking features of object-oriented programming
  - Emphasis on *data* rather than *procedure*
  - Programs are divided into what are known as *objects*
  - Data structure are designed such that they characterize the *objects*
  - *Functions* that operate on the data of an object are tied together in the data structure
  - Data is *hidden* and cannot be accessed by external functions
  - Objects may communicate with each other through functions
  - *New data* and *functions* can be easily added whenever necessary
  - Follows bottom up approach in program design

Top Features of OOPS



Object Oriented Programming Paradigm

"Object oriented programming is an approach that provides away of modularizing programs by creating partitioned memory area for both data and functions and that can be used as templates for creating copies of such modules on demand".

Basic Concepts of Object Oriented Programming

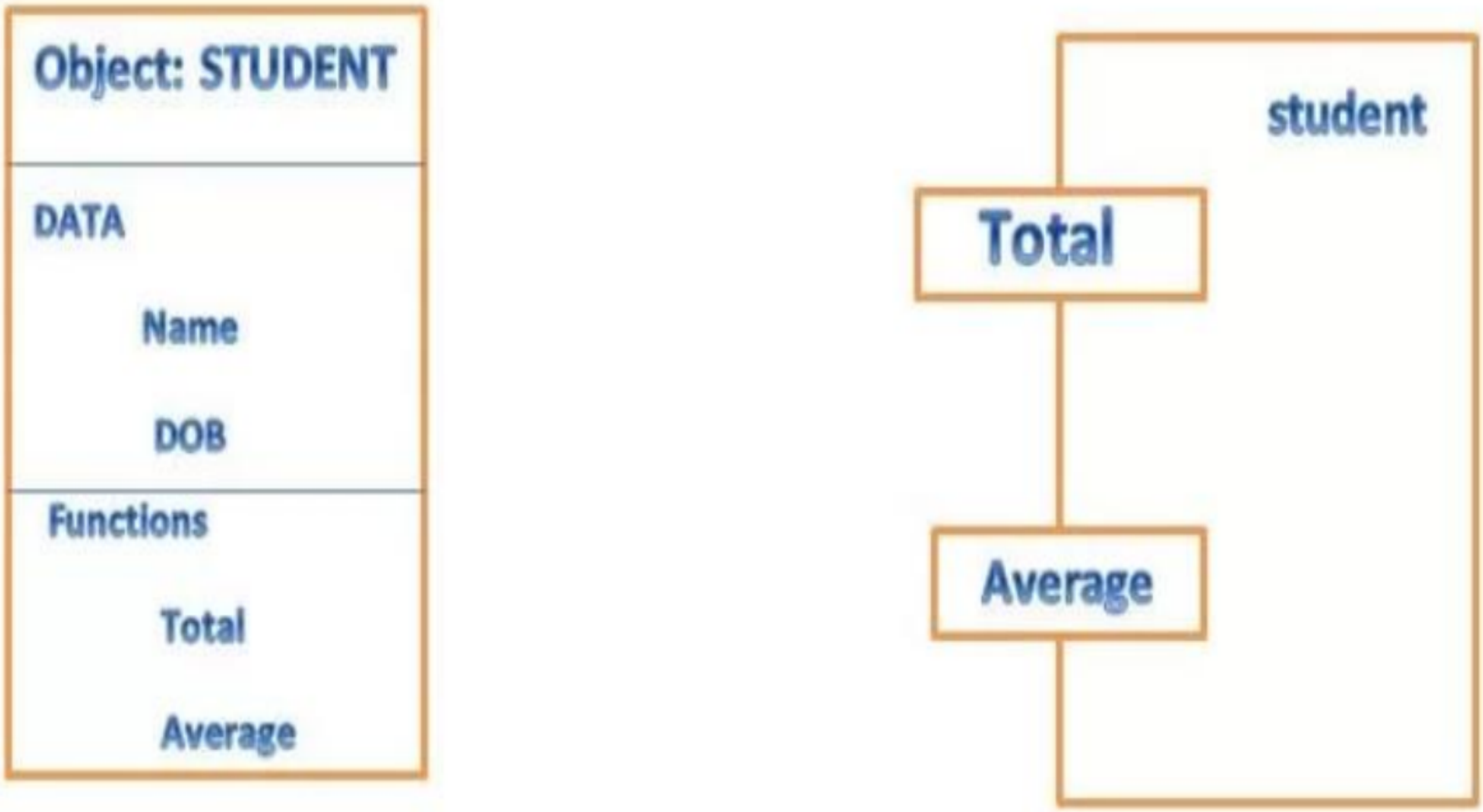
- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

What is an Object?

- Objects contain data and code to manipulate the data
- The entire set of data and code of an object can be made a *user-defined data type* with the help of a *class*
- Objects are *variable of type class*
- Objects are the *basic run time entities* in an object-oriented system.
- They may represent a person, a place, a bank account, a table of data or any item that the program has to handle
- They may also represent user-defined data such as vectors, time and lists.

Objects

- Programming problem is analyzed in term of objects and the nature of communication between them
- Program objects should be chosen such that they match closely with the real-world objects
- Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in c.
- Objects can interact without having to know details of each other's data or code



Two ways of represent of objects

Classes

- Prototype or blueprint from which objects are created
- The entire set of data and code of an object can be made a *user-defined data type* with the help of a *class*
- Objects are *variable of type class*
- Each object is associated with data of type class
- Once a class has been defined, we can create any number of objects belonging to that class
  - A class is thus a collection of objects of similar types
  - For example, *mango, apple* & *orange* are member of the class *fruit*
  - If fruit has been defined as a class, then
    - *fruit mango*; will create an *object mango* belonging to the *class fruit*.

Class & Object

Object

BMW

Car

class

Color
Weight
Speed
Model

Data members

Common features

startCar( )
changeGear( )
slowDown( )
brake( )

method

Common functionalities

Data Encapsulation

- The wrapping up of data and function into a single unit (called class) is known as encapsulation.
- Data and encapsulation is the most striking feature of a class.
- The data is not accessible to the outside world, and only those functions 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

Data Abstraction

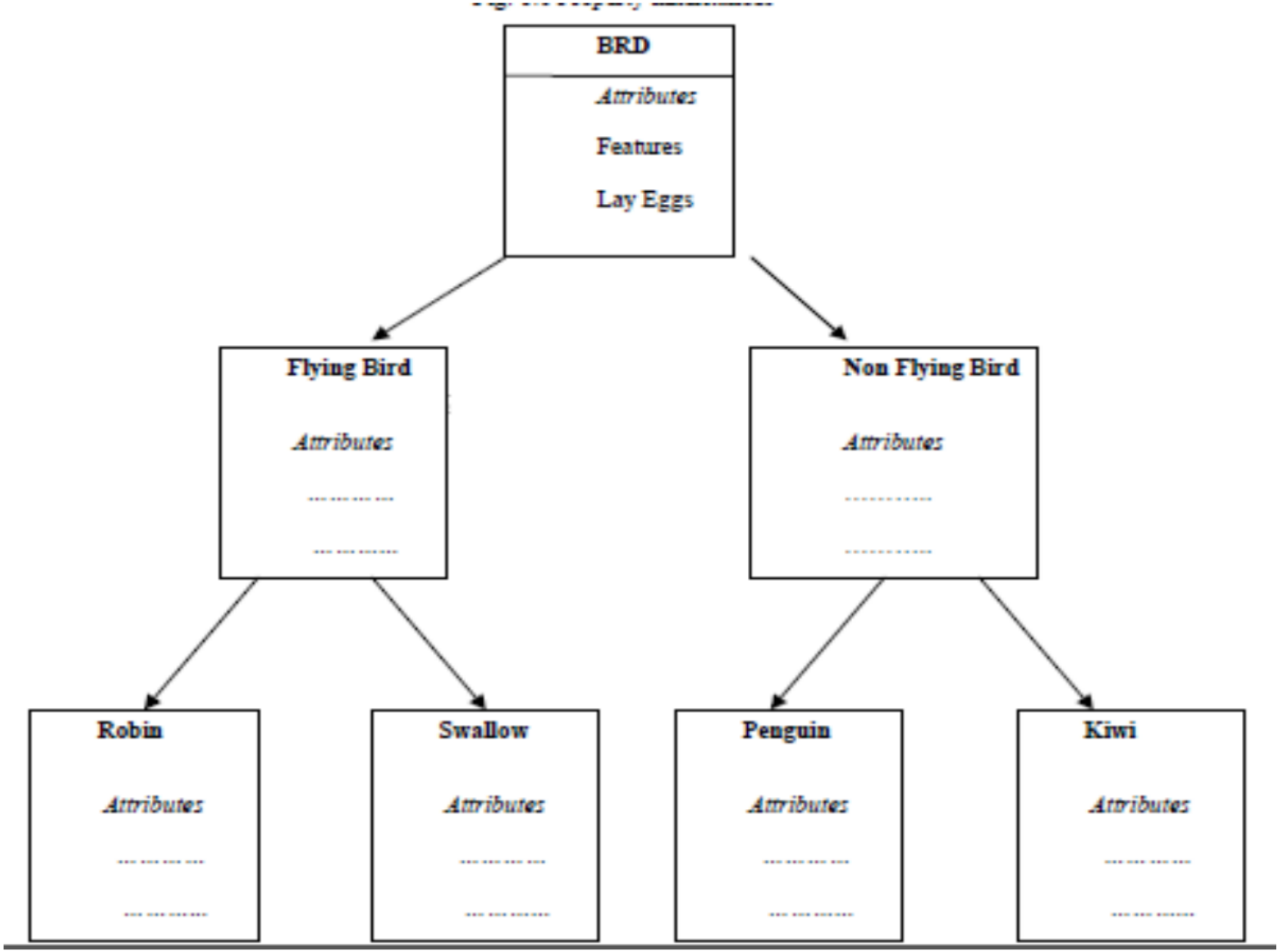
- Abstraction refers to the act of representing essential features without including the background details or explanation
- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight, and cost, and function operate on these attributes
- They encapsulate all the essential properties of the object that are to be created
- The attributes are some time called data members because they hold information.
- The functions that operate on these data are sometimes called methods or member function.
- Classes known as ***abstract data types***

Even though it performs a lot of actions it doesn't show us the process. It has hidden its process by showing only the main things like getting inputs and giving the output.

INHERITENCE

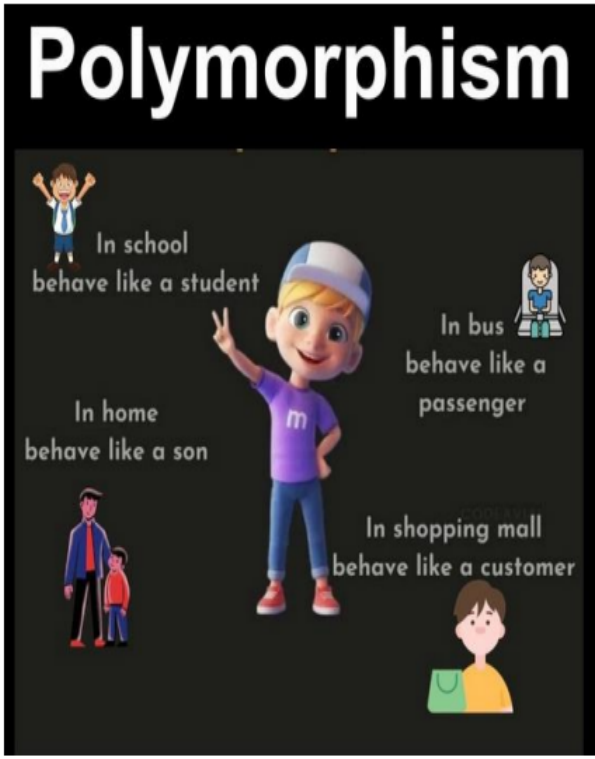
- Inheritance is the process by which objects of one class acquire the properties of another class.
- the concept of inheritance provides the idea of reusability.
- This mean that we can add additional features to an existing class with out modifying it.
- This is possible by designing a new class will have the combined features of both the classes.

Inheritance



Polymorphism

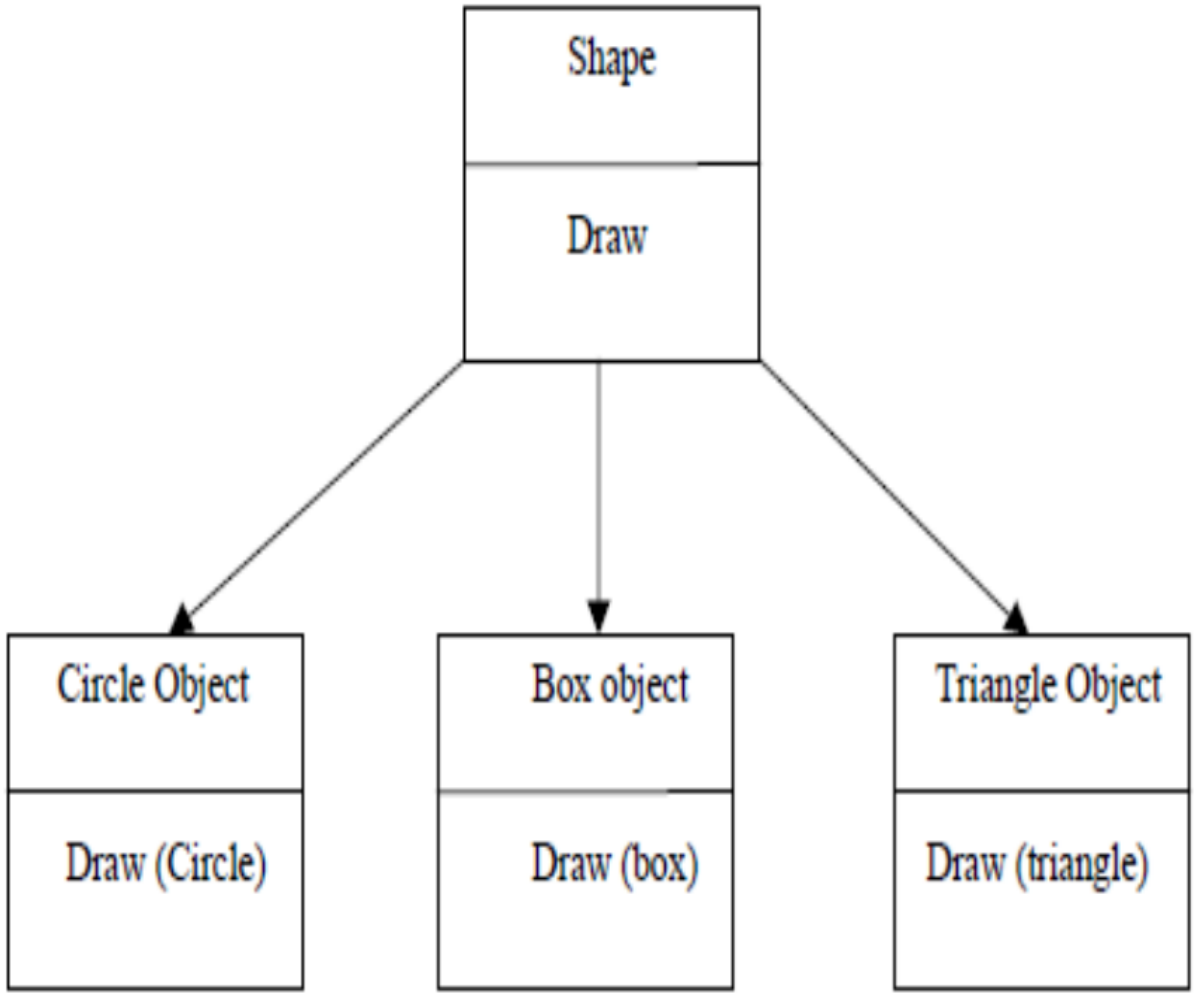
- Polymorphism is another important OOP concept.
- Polymorphism, a Greek term, means the ability to take more than on form.
- An operation may exhibit different behavior is different instances.
- The behavior depends upon the types of data used in the operation.
- For example, consider the operation of addition. For two numbers, the operation will generate a sum.
- If the operands are strings, then the operation would produce a third string by concatenation.
- The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.



Polymorphism

- A single function name can be used to handle different number and different types of argument.
- This is something similar to a particular word having several different meanings depending upon the context.
- Using a single function name to perform different type of task is known as *function overloading*.
- Polymorphism is extensively used in implementing inheritance.
- In C++, both run-time and compile-time polymorphism are supported.

Polymorphism

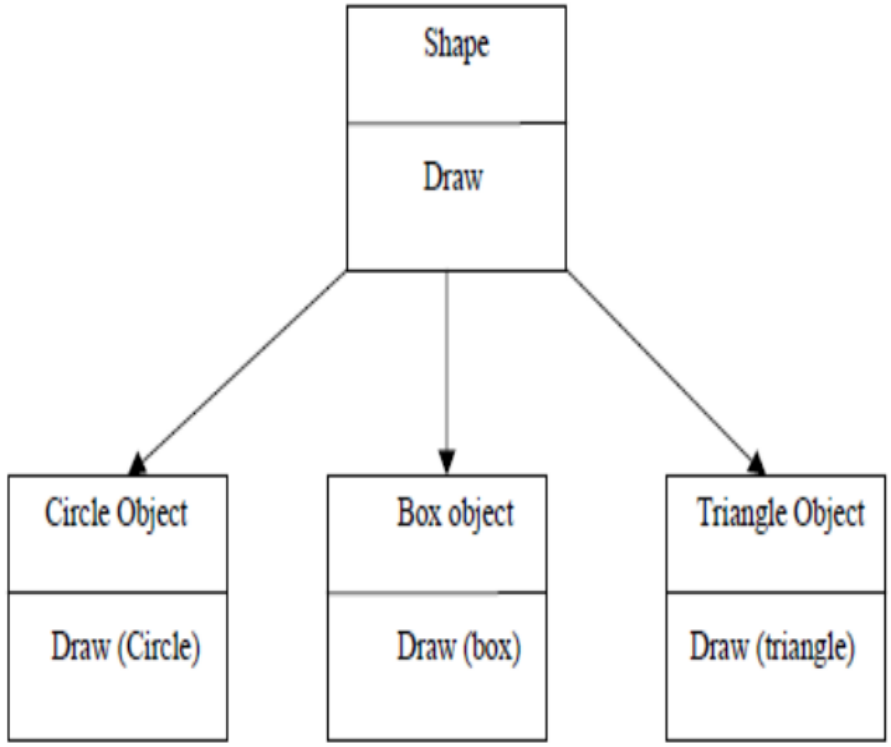


Dynamic Binding

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at **run time**.
- It is associated with polymorphism and inheritance.
- A function call associated with a polymorphic reference depends on the dynamic type of that reference.

Dynamic Binding

- Consider the procedure “draw”, by inheritance every object will have this procedure.
- Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object.
- At **run-time**, the code matching the object under current reference will be called.



Message Passing

- An object-oriented program consists of a set of objects that communicate with each other.
- The process of programming in an object-oriented language, involves the following basic steps:

1. Creating classes that define object and their behavior,
2. Creating objects from class definitions, and
3. Establishing communication among objects.

Message Passing

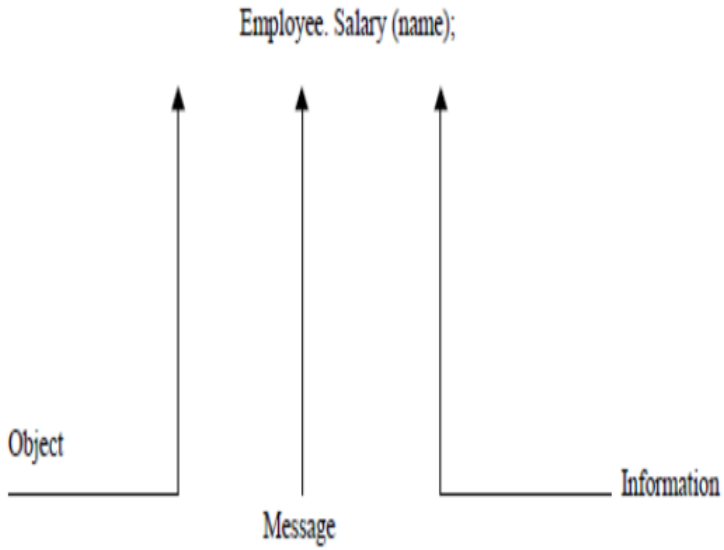
- A Message for an object is a **request for execution of a procedure**, and therefore will invoke a function (procedure) in the receiving object that generates the desired results.

- Object has a life cycle.

- They can be created and destroyed.

- Communication with an object is feasible as long as it is alive.

- *Message passing involves*
  - *specifying the name of object,*
  - *the name of the function (message) and*
  - *the information to be sent.*



Benefits of OOP

- Through **inheritance**, we can eliminate redundant code and extend the use of existing classes.

- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to **saving of development time** and **higher productivity**.

- The principle of **data hiding** helps the programmer to build secure program that can not be invaded by code in other parts of a programs.

- It is possible to have multiple instances of an object to co-exist without any interference.

- It is possible to map objects in the problem domain to those in the program.

Benefits of OOP contd.,

- It is easy to partition the work in a project based on objects.
- The **data-centered design** approach enables us to capture more detail of a model in implementable form.
- Object-oriented system can be **easily upgraded** from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Application of OOP

The promising areas for application of OOP include:

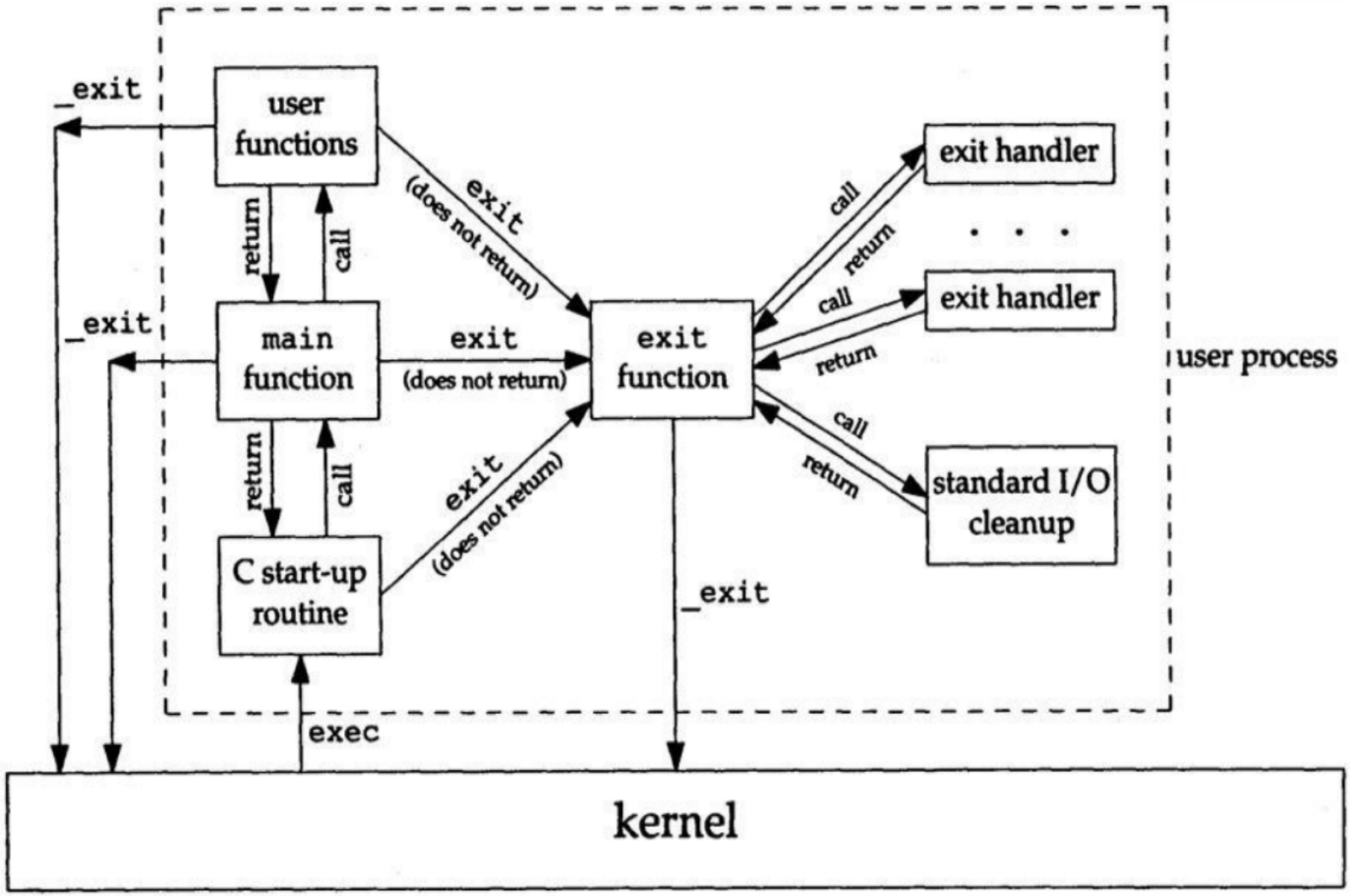
- Real-time system
- Simulation and modelling
- Object-oriented data bases
- Hypertext, Hypermedia, and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM(computer-integrated manufacturing)/CAM/CAD systems
- Arduino Integrated Development Environment (IDE)

Beginning with C++

Program feature

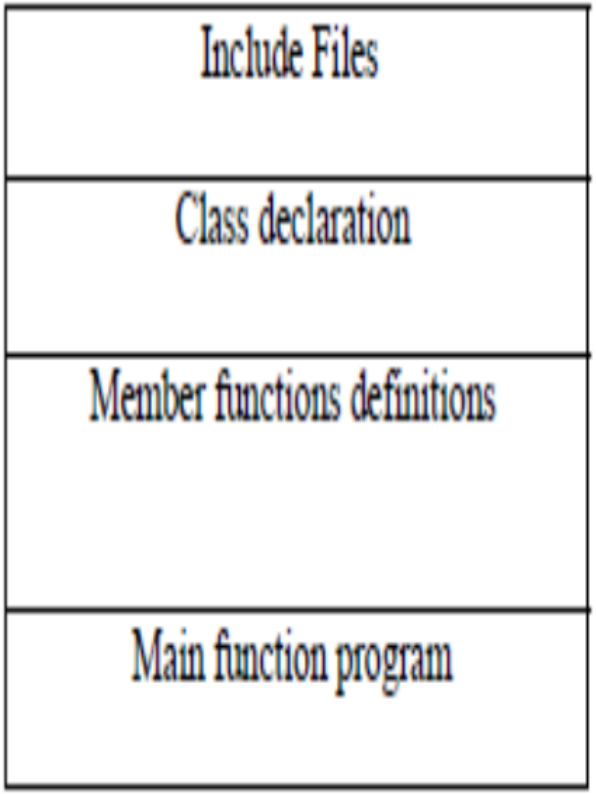
- Like C, the C++ program is a collection of functions.
- As usual execution begins at main().
- Every C++ program must have a main().
- C++ is a free form language.
- With a few exception, the compiler ignore carriage return and white spaces.
- Like C, the C++ statements terminate with semicolons

Now lets see How a C program is started and how it terminates



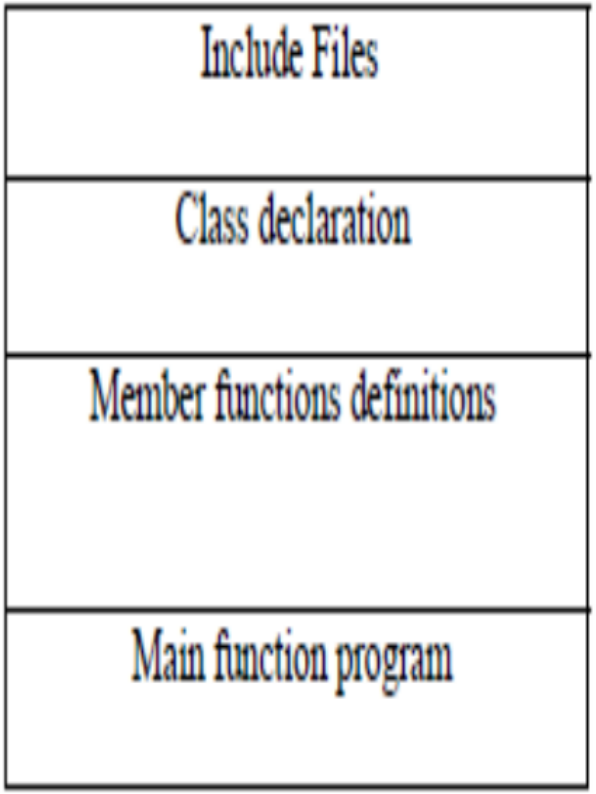
Structure of C++ Program

- A typical C++ program would contain four sections
- These sections may be placed in separate code files and then compiled independently or jointly.
- It is a common practice to organize a program into three separate files.
- The class declarations are placed in a header file and the definitions of member functions go into another file.



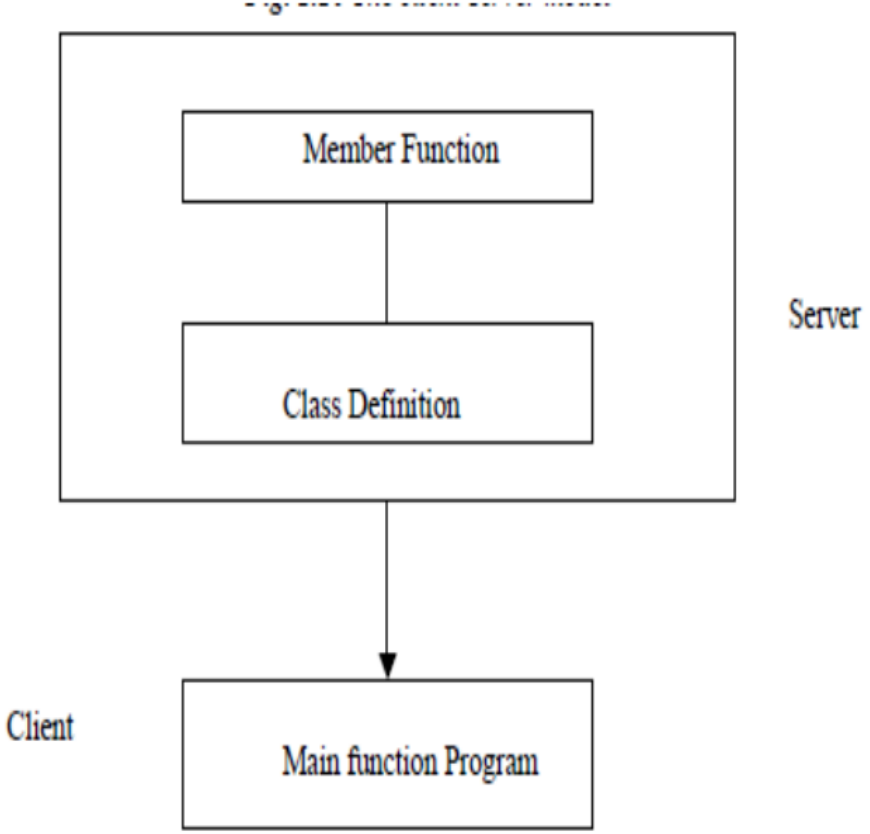
Structure of C++ Program

- This approach enables the programmer to separate the abstract specification of the interface from the implementation details (member function definition).
- Finally, the main program that uses the class is places in a third file which “includes” the previous two files as well as any other file required.



Structure of C++ Program

- This approach is based on the concept of client-server model
- The class definition including the member functions constitute the **server** that provides services to the main program known as **client**.
- The client uses the server through the public interface of the class.



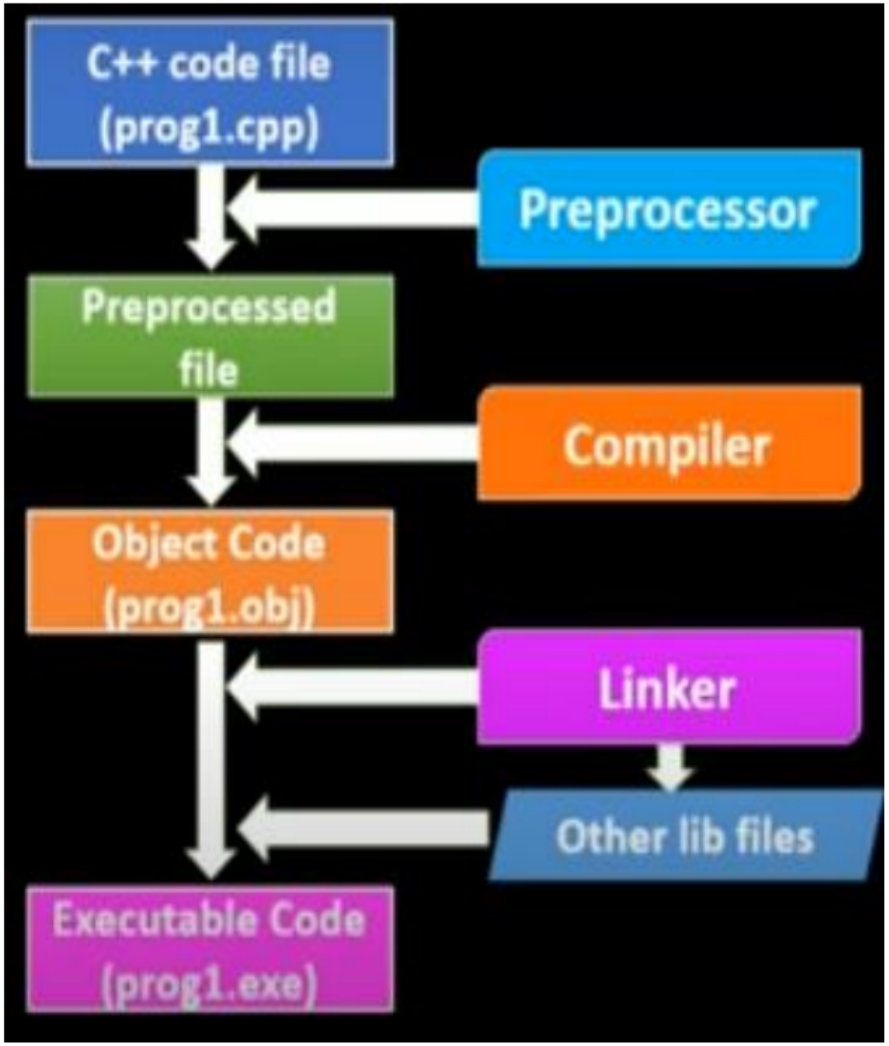
Creating, Compiling and Linking the Source File

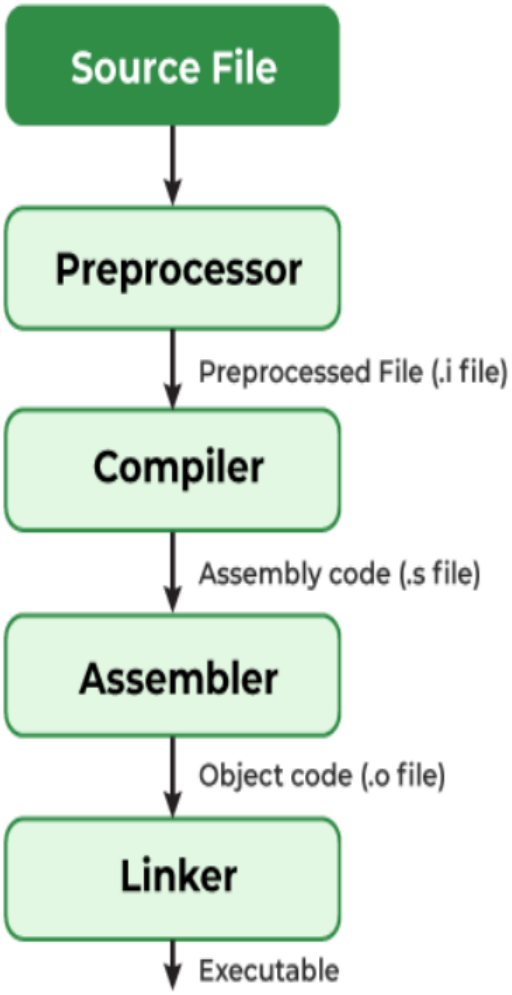
- vi, nano, ed editors used in UNIX environment.
- Windows provides integrated environment.
- Open new file and save the files.
- Save the files with ***.cpp*** extension.
- For compilation in unix ***cc filename.cpp***

  In windows ***alt+f9***
- For execution in unix ***./a.out***

  In windows ***ctl+f9***

Creating, Compiling and Linking the Source File





Comments

- The double slash comment is basically a single line comment. Multiline comments can be written as follows:

```
// This is an example of
// C++ program to illustrate
// some of its features
```

- The C comment symbols /*,*/ are still valid and are more suitable for multiline comments. The following comment is allowed:

```
/* This is an example of
C++ program to illustrate
some of its features
*/
```

Hello World

C Program

```
// FileName:HelloWorld.c:

#include <stdio.h>

int main() {
  printf("Hello World in C");
  printf("\n");
  return 0;
}
```

C++ Program

```
// FileName:HelloWorld.cpp:

#include <iostream>

int main() {
  std::cout << "Hello World in C++";
  std::cout << std::endl;
  return 0;
}
```

The #include directive instructs the compiler to include the contents of the file enclosed within angular brackets into the source file.

Hello World in C	Hello World in C++
•IO Header is stdio.h	•IO Header is iostream
•printf to *print* to console	•operator<< to *stream* to console
•Console is stdout file	•Console is std::cout ostream (in std namespace)
•printf is a variadic function	•operator<< is a binary operator
•\n to go to the new line	•std::endl (in std namespace) to go to the new line
•\n is escaped newline character	

C++ Program: Prints a Line of Text

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

Fig. 2.1 | Text-printing program.

First Program in C++: Printing a Line of Text (Cont.)

- A preprocessor directive is a message to the C++ preprocessor.
- Lines that begin with # are processed by the preprocessor before the program is compiled.
- #include <iostream> notifies the preprocessor to include in the program the contents of the input/output stream header file <iostream>.
  - Must be included for any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output.

Return Type of main()

- In C++, main () returns an integer value to the operating system.
- Therefore, every main () in C++ should end with a return (0) statement; otherwise a warning an error might occur.
- Since main () returns an integer type for main () is explicitly specified as **int.**
- **Note that the default return type for all function in C++ is int.**

Namespace

- Namespace is a new concept introduced by the ANSI C++ standards committee.
- This defines a scope for the identifiers that are used in a program.
- It provides a solution for preventing name conflicts in large projects
- A namespace allows for identically name entities as long as the namespace are different.

Understanding namespace

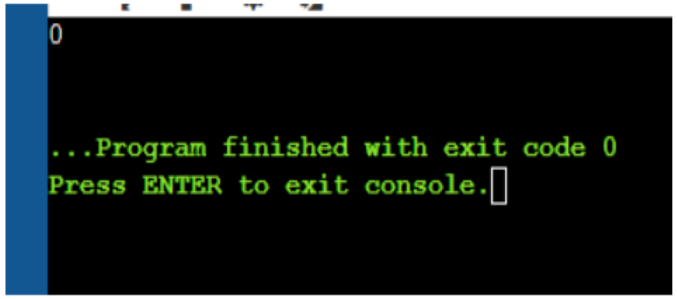
```
#include <iostream.h>
int main()
{
        int x=0;
        int x=1;
        return 0;
}
```

Compilation failed due to following error(s).

```
main.cpp: In function 'int main()':
main.cpp:14:9: error: redeclaration of 'int x'
   14 |     int x=1;
      |         ^
main.cpp:13:9: note: 'int x' previously declared here
   13 |     int x=0;
      |         ^
```

Understanding namespace

```
#include <iostream>
namespace first
{
        int x=1;
}
int main()
{
        int x=0;
        std::cout << x << std::endl;
        return 0;
}
```



Understanding namespace

```cpp
#include <iostream>

namespace first
{
    int x=1;

}

namespace sec
{
    int x=2;

}


int main()
{
    int x=0;
    std::cout << x << std::endl;
    std::cout << first::x << std::endl;
    std::cout << sec::x << std::endl;
    return 0;
}
```

```
0
1
2


...Program finished with exit code 0
Press ENTER to exit console.
```

Namespace

- In the previous code, **std** is the namespace where ANSI C++ standard class libraries are defined.
- All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope.
- **using** and **namespace** are the new keyword of C++.

Using namespace

- For using the identifier defined in the namespace scope we must include the **using** directive, like using namespace std;
- using declarations
  - eliminate the need to repeat the std:: prefix as we did in earlier programs.
- Once we insert these using declarations, we can write
  - cout instead of std::cout,
  - cin instead of std::cin and
  - endl instead of std::endl
- Many programmers prefer to use the declaration using namespace std;
  - enables a program to use all the names in any standard C++ header file (such as <iostream>) that a program might include.

Lets check??

```cpp
#include <iostream>

namespace first{
    int x = 1;
}
namespace second{
    int x = 2;
}

int main() {
    using namespace first;

    std::cout << x;

    return 0;
}
```

```cpp
#include <iostream>

namespace first{
    int x = 1;
}
namespace second{
    int x = 2;
}

int main() {
    using namespace first;

    std::cout << second::x;

    return 0;
}
```

First Program in C++: Printing a Line of Text (Cont.)

- When a cout statement executes, it sends a stream of characters to the standard output stream object
  - std::cout; normally "connected" to the screen.

- The std:: before cout is required when we use names that we've brought into the program by the preprocessor directive #include <iostream>.

- The notation std::cout specifies that we are using a name, in this case cout, that belongs to "namespace" std.

- The names cin (the standard input stream) and cerr (the standard error stream) also belong to namespace std.

Output operator

- The output statement is **cout<<"C++ is better than C.";** causes the string in quotation marks to be displayed on the screen.

- This statement introduces two new C++ features,
  cout and **<<**

- The identifier **cout** is a predefined object that represents the standard output stream in C++.

Output operator

- Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices.

- The operator << is called the **insertion** or **put to** operator.

- The << operator is referred to as the **stream insertion operator.**
  - The value to the operator's right, the right operand, is inserted in the output stream.

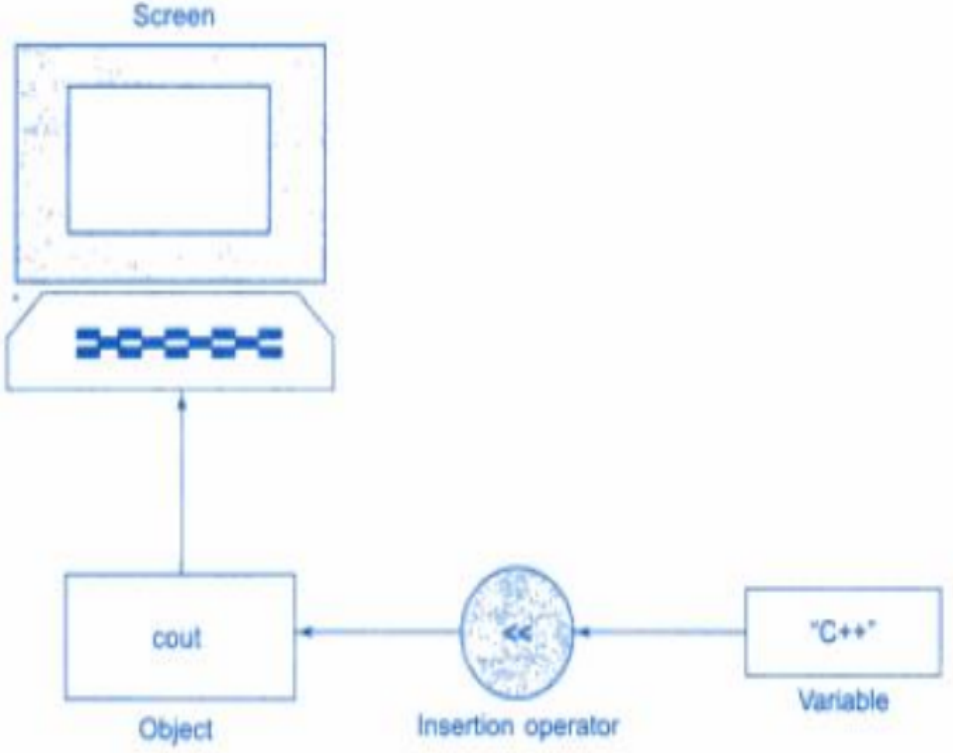
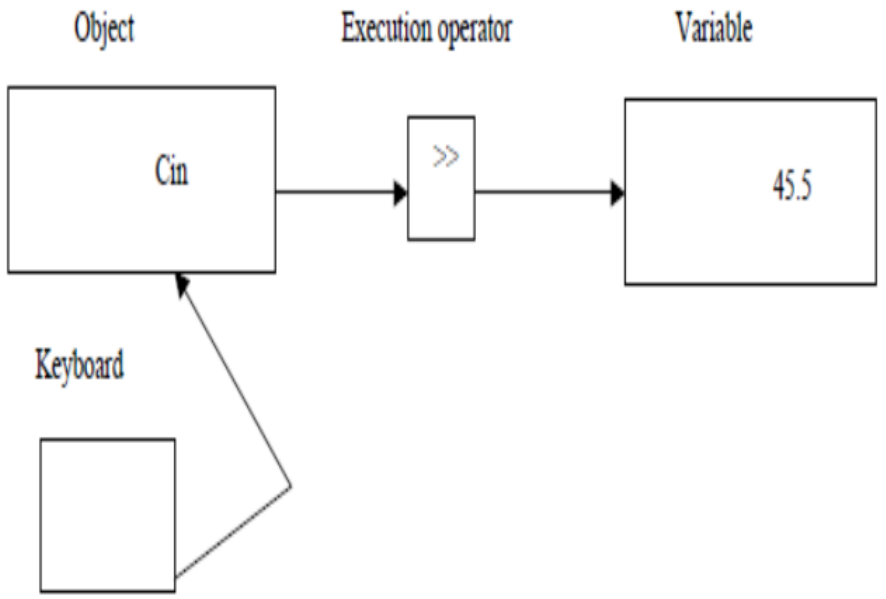


Fig. 2.1 ⇔ *Output using insertion operator*

Input Operator

- The operator >> is known as extraction or get from operator.
- It extracts (or takes) the value from the keyboard and assigns it to the variable on its right
- This corresponds to a familiar scanf() operation.
- Like <<, the operator >> can also be overloaded.



1.8 Input using extraction operator

Input Operator

- A prompt directs the user to take a specific action.
- A cin statement uses the input stream object cin (of namespace std)
  - the stream extraction operator, >>, is used to obtain a value from the keyboard.
- Using the stream extraction operator with std::cin takes character input from the standard input stream (the keyboard).

C++ Program: Adding Integers

```
1  // Fig. 2.5: fig02_05.cpp
2  // Addition program that displays the sum of two integers.
3  #include <iostream> // allows program to perform input and output
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // prompt user for data
14    std::cin >> number1; // read first integer from user into number1
15
16    std::cout << "Enter second integer: "; // prompt user for data
17    std::cin >> number2; // read second integer from user into number2
18
19    sum = number1 + number2; // add the numbers; store result in sum
20
21    std::cout << "Sum is " << sum << std::endl; // display sum; end line
22 } // end function main
```

Another C++ Program: Adding Integers (Cont.)

- std::endl is a so-called stream manipulator.

- The name endl is an abbreviation for “end line”
  - belongs to namespace std.

- The std::endl stream manipulator outputs a newline, then “flushes the output buffer.”
  - This simply means that, on some systems where outputs accumulate in the machine until there are enough to “make it worthwhile” to display them on the screen, std::endl forces any accumulated outputs to be displayed at that moment.
  - This can be important when the outputs are prompting the user for an action, such as entering data.

Another C++ Program: Adding Integers (Cont.)

- Using multiple stream insertion operators (<<) in a single statement is referred to as concatenating, chaining or cascading stream insertion operations.
- Calculations can also be performed in output statements.

Program: Add two numbers

C Program	C++ Program

// FileName:Add_Num.c:
#include <stdio.h>
int main() {
int a, b; int sum;
 	
printf("Input two numbers:\n"); scanf("%d%d", &a, &b);
 	
sum = a + b;
 	
printf("Sum of %d and %d", a, b);
printf(" is: %d\n", sum);
return 0;
}
	// FileName:Add_Num_c++.cpp:
	#include <iostream>
	int main() {
	int a, b;
	
	std::cout << "Input two numbers:\n";
	std::cin >> a >> b;
	int sum = a + b; // Declaration of sum
	std::cout << "Sum of "<< a << " and "<< b << " is:" << sum <<
	std::endl;
	return 0; }

Input two numbers: 3 4 Sum of 3 and 4 is: 7	Input two numbers: 3 4 Sum of 3 and 4 is: 7
• scanf to *scan* (*read*) from console • Console is stdin file • scanf is a variadic function • Addresses of a and b needed in scanf • All variables a, b & sum declared first (C89) • Formatting (%d) needed for variables	• operator>> to *stream* from console • Console is std::cin istream (in std namespace) • operator>> is a binary operator • a and b can be directly used in operator>> operator • sum may be declared when needed • Formatting is derived from type (int) of variables

C Standard Library	C++ Standard Library
•All names are global •stdout, stdin, printf, scanf	•All names are within std namespace •std::cout, std::cin •Use using namespace std; to get rid of writing std:: for every standard library name

W/o using

```
#include <iostream>
int main( ) {
   std::cout << "Hello World
      in C++"<< std::endl;

   return 0;
}
```

W/ using

```
#include <iostream>
using namespace std;
int main( ) {
   cout << "Hello World in C++"<< endl;
   return 0;
 }
```

Write a program to find sum of n natural numbers

Program : Sum n natural numbers

C Program	C++ Program

// FileName:Sum_n.c:
#include <stdio.h>
 	
int main() { int n; int i;
int sum = 0;
 	
printf("Input limit:\n");
scanf("%d", &n);
 	
for (i = 0; i <= n; ++i)
sum = sum + i;
printf("Sum of %d", n);
printf(" numbers is: %d\n", sum);
return 0;
}
	// FileName:Sum_n_c++.cpp:
	#include <iostream>
	using namespace std;
	
	int main() { int n;
	int sum = 0;
	
	cout << "Input limit:" << endl;
	cin >> n;
	for (int i = 0; i <= n; ++i) // Local Decl
	sum = sum + i;
	cout << "Sum of " << n ;
	cout << " numbers is: " << sum << endl;
	return 0;
	}

Input limit: 10 Sum of 10 numbers is: 55	Input limit: 10 Sum of 10 numbers is: 55
•i must be declared at the beginning	•i declared locally in for loop

Tokens

- The smallest individual units in a program are known as tokens.
- C++ has the following tokens
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - operators

Another C++ Program

```
16     cout << "Enter two integers to compare: "; // prompt user for data
17     cin >> number1 >> number2; // read two integers from user
18
19     if ( number1 == number2 )
20        cout << number1 << " == " << number2 << endl;
21
22     if ( number1 != number2 )
23        cout << number1 << " != " << number2 << endl;
24
25     if ( number1 < number2 )
26        cout << number1 << " < " << number2 << endl;
27
28     if ( number1 > number2 )
29        cout << number1 << " > " << number2 << endl;
30
31     if ( number1 <= number2 )
32        cout << number1 << " <= " << number2 << endl;
33
34     if ( number1 >= number2 )
35        cout << number1 << " >= " << number2 << endl;
36  } // end function main
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)

Conclusion

- What is an object-oriented programming?
- Using namespace
- A basic program in C++ using `cin` and `cout`

- Write a C++ program that swaps the values of two variables without using a temporary variable.

- Write a C++ program that checks if a given year is a leap year and also centenary year

- Write a C++ program to check if a given number is prime.

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

int main() {
    int a = 5, b = 10;

    // Before swapping
    cout << "Before swap: a = " << a << ", b = " << b << endl;

    // Swap without using a temporary variable
    a = a + b;  // a now holds the sum of a and b
    b = a - b;  // b is now the original value of a
    a = a - b;  // a is now the original value of b

    // After swapping
    cout << "After swap: a = " << a << ", b = " << b << endl;

    return 0;
}
```

CLASS & OBJECTS

CLASS

- In C++, a class is a new data type that contains member variables and member functions that operates on the variables.

- It allows the data to be hidden, if necessary from external use.

- When we defining a class, we are creating a new abstract data type that can be treated like any other built in data type.

Generally a class specification has two parts:-

a) Class declaration - describes the type and scope of its members.

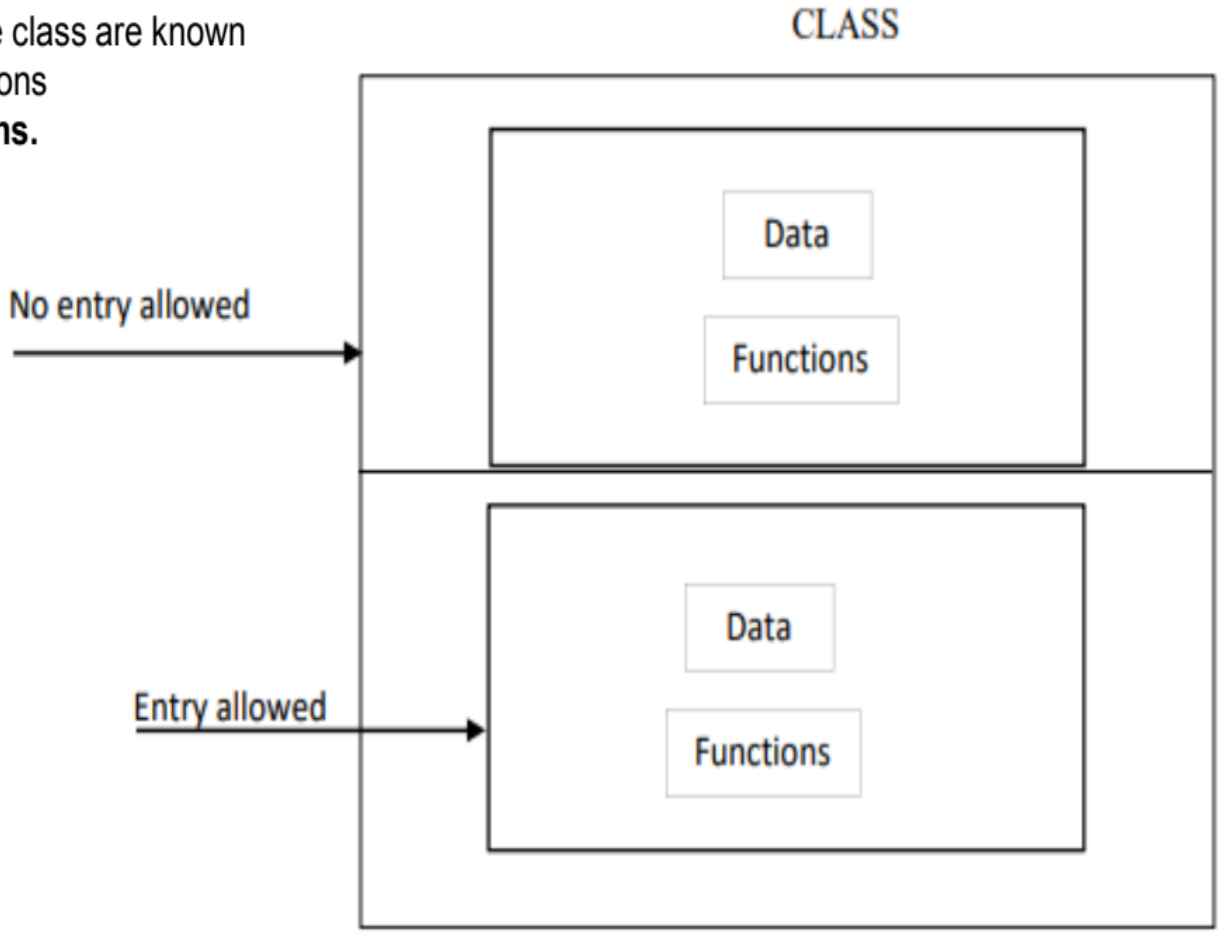
b) Class function definition - describes how the class functions are implemented.

Class Declaration:

```
class class-name
{
    private:
        variable declarations;
        function declaration ;
    public:
        variable declarations;
        function declaration;
};
```

Data hiding in classes:

The variables declared inside the class are known as **data members** and the functions are known as **members functions.**



```
class product
{
    int count; // declaration of variables is by default private
    float price;

    public:
        void getdata(int x, float y); // declaration of functions through prototype
        void putdata(void);
};
```

Objects

```
product m;  //memory is created for m

product m, n, o;

class product
{       ……..
    ……..
    ……..
} m, n, o;
```

Accessing Class Members

object-name.function-name (actual-arguments);

m.getdata(50,85.7);

If a variable is declared as public, it can be accessed directly by an object

```
class abc
{
    int m;
    int n;
    public:
        int p;
};
……
……
abc z;
z.m = 0; // m is private. Error in statement
z.p = 5; // p is public. Statement is OK ……. …….
```

DEFINING MEMBER FUNCTION:

Member can be defined in two places

- Outside the class definition
- Inside the class function

OUTSIDE THE CLASS DEFINITION

Syntax:

```
return type class-name::function-name(argument declaration )
{
function-body
}
```

OUTSIDE THE CLASS DEFINITION

```
void product :: getdata(int x, float y)
{
    count = x;
    price = y;
}
void product :: putdata(void)
{
    cout << "Count :" << count << "\n";
    cout << "Price :" << price << "\n";
}
```

Some characteristics:

- Several different classes can use the same function name. The "membership label" will resolve their scope.
- Member functions can access the private data of the class. A non member function can't do so. (Exception is *friend* function)
- A member function can call another member function directly, without using the dot operator.

INSIDE THE CLASS DEFINITION:

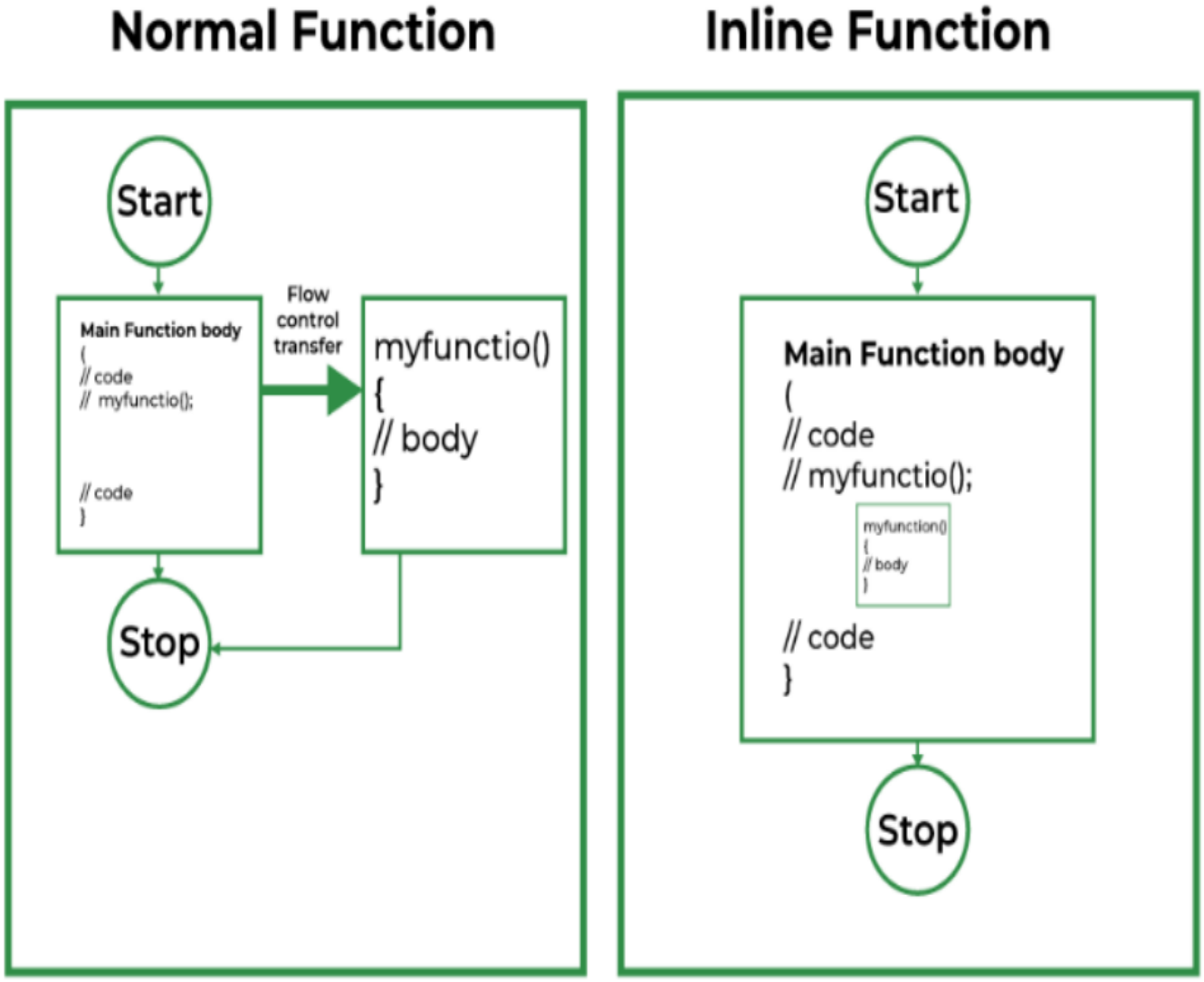
```
class product
{
        int count;
        float price;
        public:
                void getdata(int x, float y); // a function declaration
                // following is an inline function

                void putdata(void) // a function definition within class
                {
                        cout << count << "\n";
                        cout << price << "\n";
                }
};
```

A C++ Program with Class:

Write a simple program using class in C++ to input subject mark and prints it.

Inline Functions



When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call.

This substitution is performed by the C++ compiler at compile time.

An inline function may increase efficiency if it is small.

```
inline function-header
{
   function body
}
```

Inline Functions

- One of the major characteristics of Object Oriented Programming is isolating the implementation details from the class definition.
- Defining member functions outside a class is therefore a good practice.
- We can make a member function inline even if it is defined outside the class.
- For this we just have to use the qualifier inline in the header of the function definition.

Making an Outside Function Inline

```
class product
{
      ……
      ……
      public:
            void getdata(int x, float y); // function declaration
};
inline void product :: getdata(int x, float y) // function definition outside the class
{
      count = x;
      price = y;
}
```

Nesting of Member Functions

```
#include<iostream>
using namespace std;
class value
{
        int a,b;
    public:
        void enter(void);
        void output(void);
        int biggest(void);
};
int value :: biggest (void)
{
        if(a >= b)
                return (a);
        else
                return (b);
}
void value :: enter(void)
{
        cout << " Enter the values of a and b" << "\n";
        cin >> a >> b;
}
void value :: output(void)
{
        cout << "Biggest value is " << biggest() << "\n"; // calling member function
}
int main()
{
        value M;
        M.enter();
        M.output();
        return 0;
}
```

Private member functions:

```
class model
{
    int x;
    void read(void); // private member function
    public:
        void update(void);
        void write(void);
};
```

```
m.read(); // does not work
```

But the function update() can invoke the function read() to modify the value of x.

```
void model :: update(void)
{
    read(); // no object used; simple call
}
```

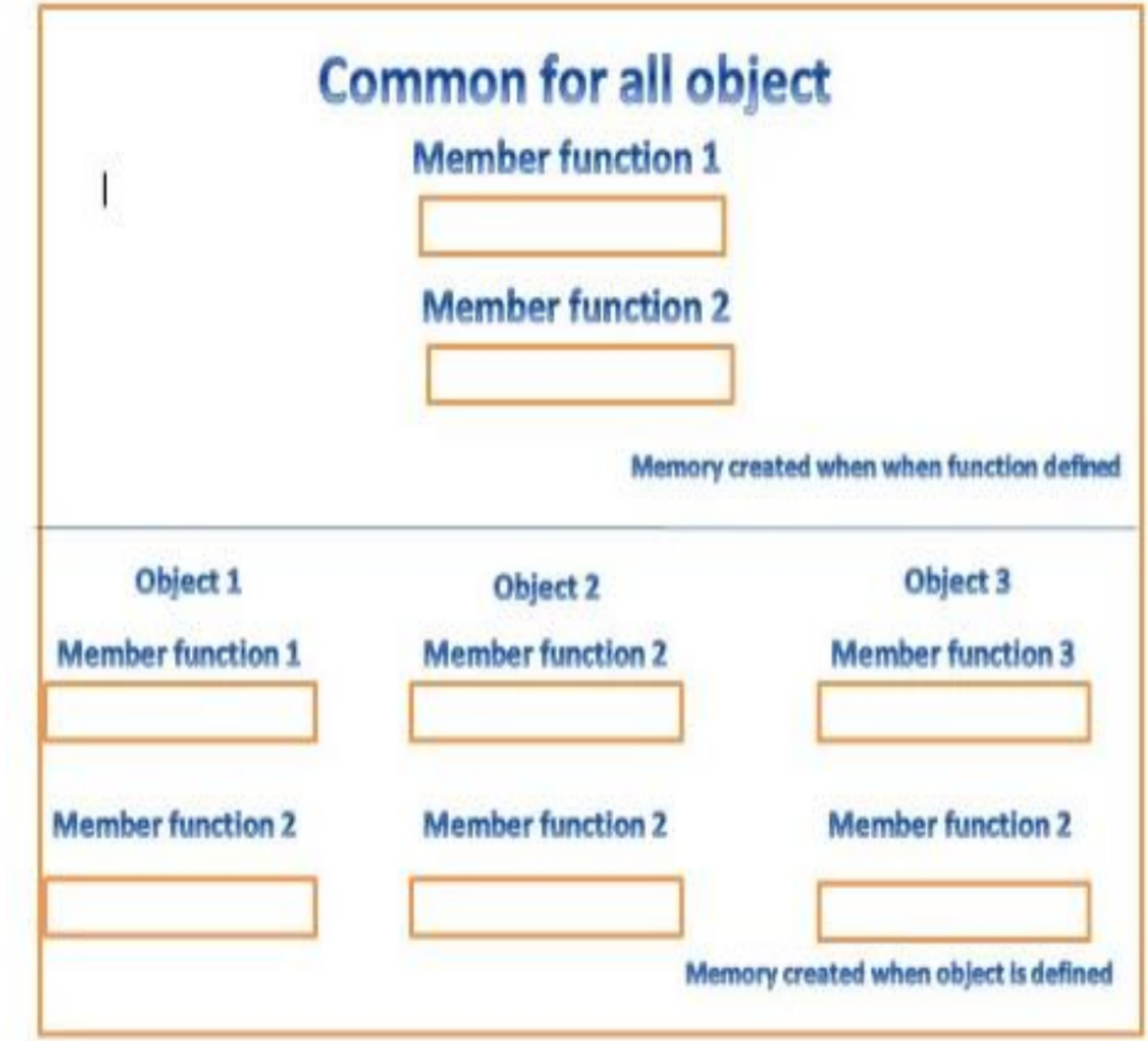
ARRAYS WITHIN A CLASS:

Create a class employee with *name* as character array, *age* and *sal* as integer variable. Implement the functions *getdata()* and *putdata()* to read and print the values for 5 employees.

(hint: create employee objects as *employee emp[5];*)

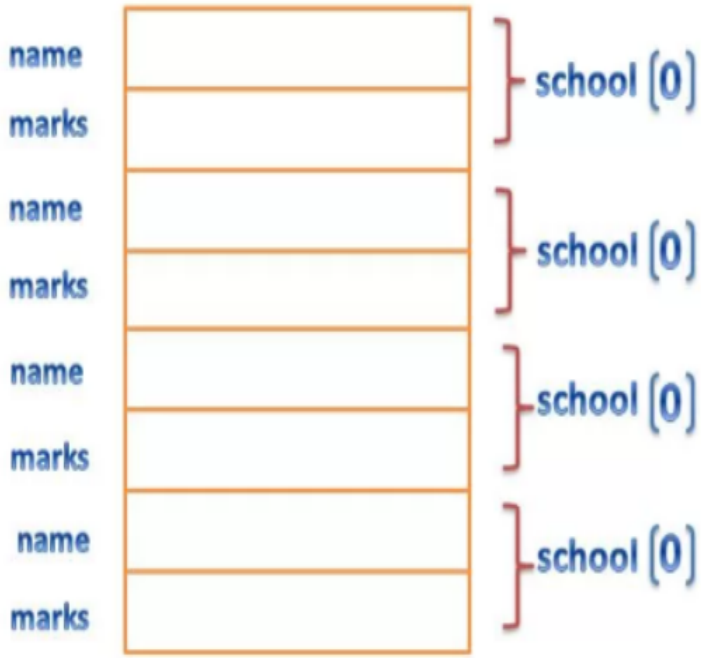
C++ Memory allocation for objects

- Memory space for objects is allocated when they are declared and not when the class is specified.
- member functions are created and placed in the memory space only once when they are defined as a part of a class specification
- whenever the objects are created no separate space is allocated for member functions
- only space for member variables is allocated separately for each object.


Common for all object
Member function 1
Member function 2
Memory created when when function defined
Object 1
Object 2
Object 3
Member function 1
Member function 2
Member function 3
Member function 2
Member function 2
Member function 2
Memory created when object is defined

C++ Arrays of Objects

```
class student
{
        char name[20];
        float marks;
public:
        void getdata();
        void putdata();
};
```



Storage of data items of an object array

```
student school[4]; //array of school
student graduate[8]; //array of graduate
student postgraduate[15]; //array of postgraduate
```