UNIT 1

Introducing Object-Oriented Programming (OOP)

BCA-2nd Sem

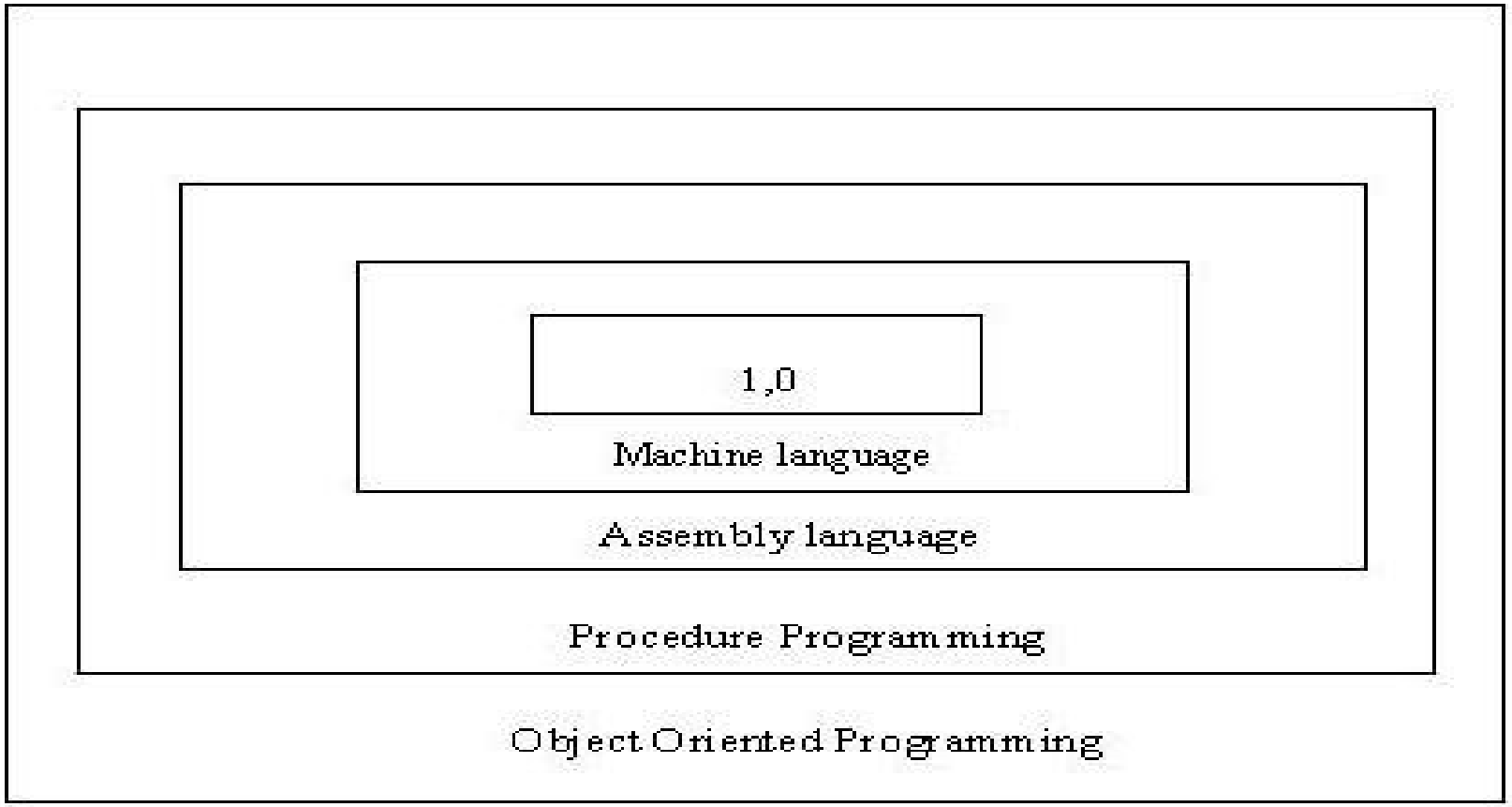
The following issued need to be addressed to face the crisis:

- How to represent real-life entities of problems in system design?
- How to design system with open interfaces?
- How to ensure reusability and extensibility of modules?
- How to develop modules that are tolerant of any changes in future?

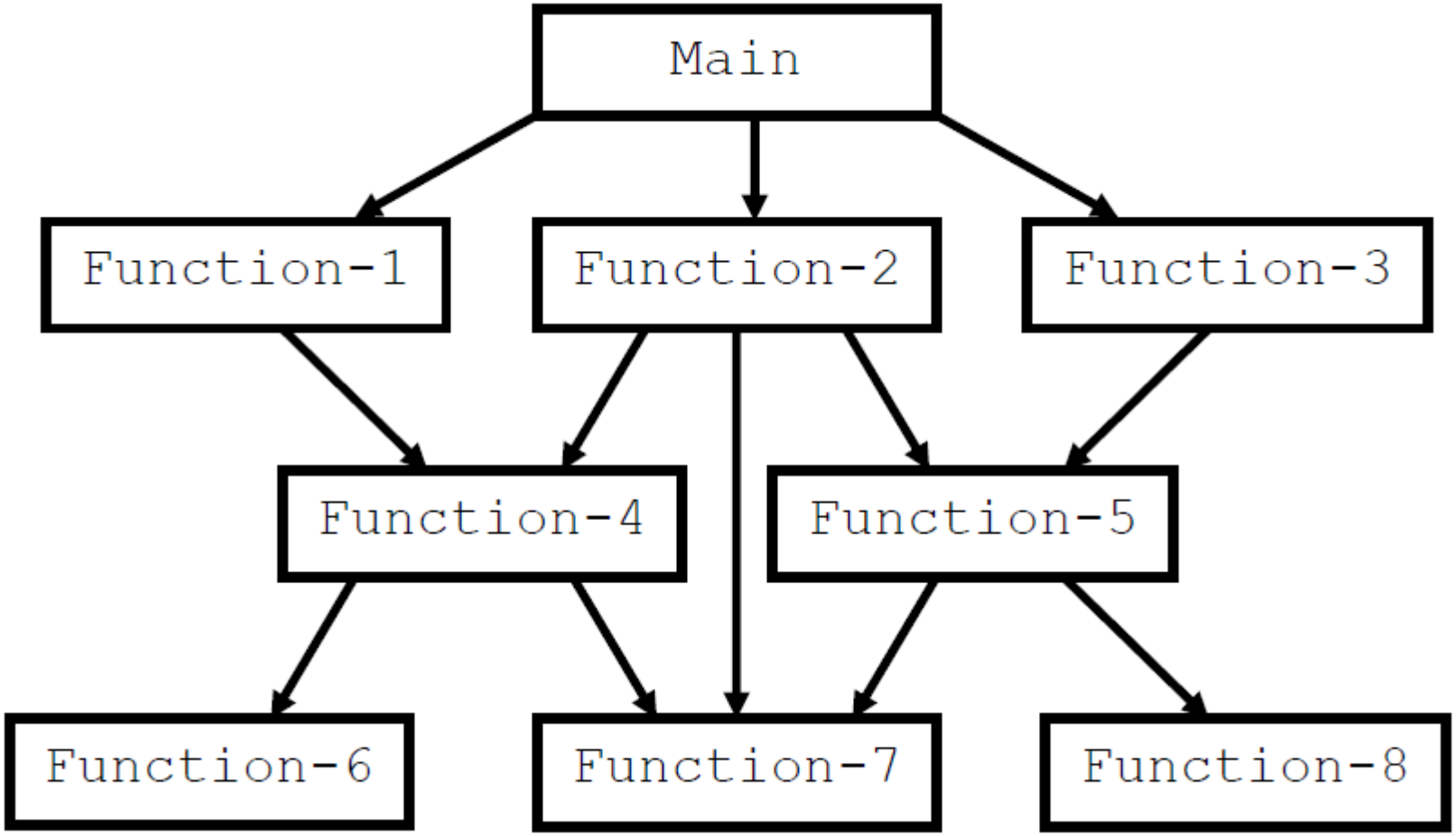
The following issued need to be addressed to face the crisis:

- How to improve software productivity and decrease software cost?
- How to improve the quality of software?
- How to manage time schedules?

Software Evolution
Layers Of Computer Software



Procedure-Oriented Programming



Organization Of Data And Function

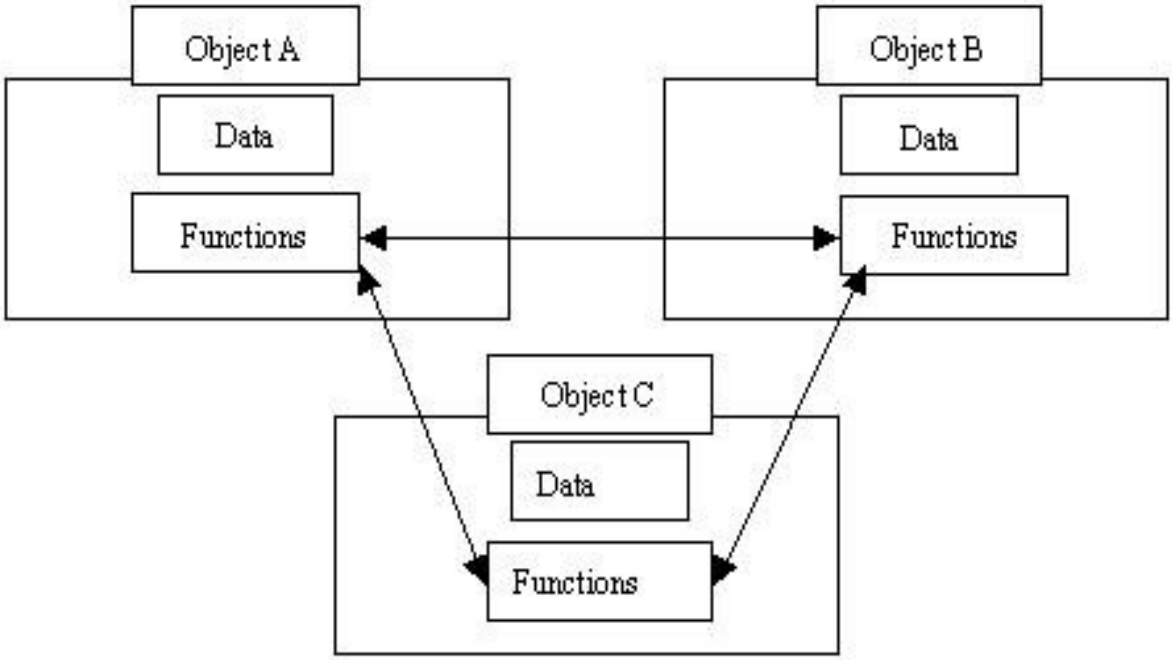


Fig 4

Characteristics of Procedure-Oriented Programming

- 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 the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design

Basic Concepts of Object Oriented Programming

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

Object Representation

OBJECTS:
STUDENT
DATA
Name
Date-of-birth
Marks
FUNCTIONS
Total
Average
Display

Classes

- A class is thus a collection of objects similar types.
- For examples, Mango, Apple and orange members of class fruit.

- Fruit Mango;
- Will create an object **mango belonging to the class fruit.**

Data Abstraction and Encapsulation

- The wrapping up of data and function into a single unit (called class) is known as encapsulation.

- The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

Data Abstraction and Encapsulation

- These functions provide the interface between the object's data and the program.

- Abstraction refers to the act of representing essential features without including the background details or explanation

Inheritance

- Inheritance is the process by which objects of one class acquired the properties of objects of another classes.

Polymorphism

- Polymorphism 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.

Polymorphism

- 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 type of task is known as function overloading.

Dynamic Binding

- 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.

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:

Message Passing

1.Creating classes that define object and their behavior,

2. Creating objects from class definitions, and

3. Establishing communication among objects.

Benefits of OOP

- Through inheritance, we can eliminate redundant code 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.

Benefits of OOP

- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- 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 can implemental form.
- Object-oriented system can be easily upgraded from small to large system.
- • Software complexity can be easily managed.

Application of OOP

- Real-time system
- Simulation and modeling
- Object-oriented data bases
- Hypertext, Hypermedia, and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

Difference Between C and C++

C	C++
1.C is Procedural Language.	**1.** C++ is non Procedural i.e Object oriented Language.
2. No virtual Functions are present in C	**2.** The concept of virtual Functions are used in C++.
3. In C, Polymorphism is not possible.	**3.** The concept of polymorphism is used in C++. Polymorphism is the most Important Feature of OOPS.
4. Operator overloading is not possible in C.	**4.** Operator overloading is one of the greatest Feature of C++.
5. Top down approach is used in Program Design.	**5.** Bottom up approach adopted in Program Design.
6. No namespace Feature is present in C Language.	**6.** Namespace Feature is present in C++ foravoiding Name collision.

Difference Between C and C++

C	C++
7. Multiple Declaration of global variables are allowed.	**7.** Multiple Declaration of global varioables are not allowed.
•**8.** In C •scanf() Function used for Input. •printf() Function used for output.	•**8.** In C++ Cin>> Function used for Input. •Cout<< Function used for output.
9. Mapping between Data and Function is difficult and complicated.	**9.** Mapping between Data and Function can be used using "Objects"
10. In C, we can call main() Function through other Functions	**10.** In C++, we cannot call main() Function through other functions.
11. C requires all the variables to be defined at the starting of a scope.	**11.** C++ allows the declaration of variable anywhere in the scope i.e at time of its Firstuse.
12. No inheritance is possible in C.	**12.** Inheritance is possible in C++

C++ Keywords

asm	Insert an assembly instruction
auto	declare a local variable
bool	declare a boolean variable
break	break out of a loop
case	part of a switch statement
catch	handles thrown exceptions
char	declare a character variable
class	declare a class
const	declare immutable data
const_cast	cast from const variables
continue	bypass iterations of a loop

C++ Keywords

default	default handler in a case statement
delete	free memory
do	looping construct
double	declare a double precision floating-point variable
dynamic_cast	perform runtime casts
else	alternate case for an if statement
enum	create enumeration types
explicit	only use constructors when they exactly match
extern	tell the compiler about variables

Header file in C++

<cstdlib>	General purpose utilities: program control, dynamic memory allocation, random numbers, sort and search
<csignal>	Functions and macro constants for sign
<csetjmp>	Macro (and function) that saves (and ju
<cstdarg>	Handling of variable length argument l
<typeinfo>	Runtime type information utilities
<typeindex> (since C++11)	std::type_index
<type_traits> (since C++11)	Compile-time type information

Header file in C++

<bitset>	std::bitset class template
<functional>	Function objects, designed for use with the standard algorithms
<utility>	Various utility components
<ctime>	C-style time/date utilites
<chrono> (since C++11)	C++ time utilites
<cstddef>	typedefs for types such as size_t, NULL and others
<initializer_list> (since C++11)	std::initializer_list class template
<tuple> (since C++11)	std::tuple class template

Comment in C++

```
int c;
/**
* compares (XOR) two Types
* return boolean result
 */
 bool compare(Type l, Type r);
```

Variables in C++

Charater type

	Character types	**char**
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	Can represent the largest supported character set.	
	Integer types (signed)	**signed char**
wchar_t	*signed* **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.

Reference Variables in C++

The declaration of the form:

<Type> & <Name>

where <Type> is type and <Name> isan identifier

whose type is **reference to <Type>**.

Reference Variables in C++

Examples:

```
int A = 5;
int& rA = A;
extern int& rB;
int& foo ();
void bar (int& rP);
class MyClass { int& m_b; /* ... */ };
int funcX() { return 42 ; }; int (&xFunc)() = funcX;
const int& ref = 65;
```

Bool Data Type

- The Boolean data type is used to declare a variablewhose value will be set as true (1) or false (0).

- To declare such a value, you use the **bool** keyword. The variable can then be initialized with the starting value.

Bool Data Type

- A Boolean constant is used to check the state of a variable, an expression, or a function, as true or false.

- **Example**:
- boolGotThePassingGrade=true;

-

Importance of function prototyping in C++.

```
#include <stdio.h>
int fac(int n);                          /* Prototype */
int main(void) {                         /* Calling function */
printf("%d\n", fac());       /* ERROR: fac is missing an
   argument! */
return 0; }
int fac(int n) {                         /* Called function */
if (n == 0)
return 1;
else
return n * fac(n - 1); }
```

Function Overloading

- Function overloading means two or more functions can have the same name but either the number of arguments or the data type of arguments has to be different.

Function Overloading

```
#include <iostream>
 int volume(int s)
 {
return s*s*s;
}
 double volume(double r, int h)
{
return 3.14*r*r*static_cast<double>(h);
 }
```

Function Overloading

```
long volume(long l, int b, int h)
{
return l*b*h;
}
int main()
 {
std::cout << volume(10);
std::cout << volume(2.5, 8);
std::cout<<volume(100,75,15);
```

Default Argumernt

- the programmer to specify default arguments that always have a value, even if one is not specified when calling the function.

- For example, in the following function declaration:

```
int my_func(int a, int b, int c=12);
result = my_func(1, 2, 3);
result = my_func(1, 2);
```

Inline Function

```
#include <iostream>
using namespace std;
 inline void hello()
 {
cout<<"hello";
}
int main() {
hello(); //Call it like a normal function... cin.get();
}
```

Scope Resolution Operator

```
#include <iostream>
 using namespace std;
 char c = 'a';
 int main()
 {
 char c = 'b';
 cout << "Local c: " << c << "\n";
 cout << "Global c: " << ::c << "\n";
 return 0;
}
```

Acess Specifier

- There are 3 access specifiers for a class/struct/Union in C++.

- These access specifiers define how the members of the class can be accessed.

- Any member of a class is accessible within that class(Inside any member function of that same class).

- Moving ahead to type of access specifiers, they are:

Acess Specifier

An Source Code Example:

```
class MyClass{
public: int a;
protected: int b;
private: int c; };
int main()
{
 MyClass obj;
 obj.a = 10;
 obj.b = 20;
 obj.c = 30;
```

Acess Specifier

- **Public -** The members declared as Public are accessible from outside the Class through an object of the class.
- **Protected -** The members declared as Protected are accessible from outside the class **BUT** only in a class derived from it.
- **Private -** These members are only accessible from within the class. No outside Access is allowed.

REFRENCES

- Learn Programming in C++ By Anshuman Sharma, Anurag Gupta, Dr.Hardeep Singh, Vikram Sharma