Object Oriented Programming (PC-CS 203A)

UNIT-1

Prepared By:

Er. Tanu Sharma

(A.P. CSE Deptt., PIET)

Syllabus (Unit-1)

Introduction to C++, C++ Standard Library, Illustrative Simple C++ Programs. Header Files, Namespaces, Application of object oriented programming.

Object Oriented Concepts, Introduction to Objects and Object Oriented Programming, Encapsulation, Polymorphism, Overloading, Inheritance, Abstract Classes, Accessifier (public/ protected/ private), Class Scope and Accessing Class Members, Controlling Access Function, Constant, Class Member, Structure and Class

Introduction to C++

- C++ is an Object Oriented Language.
- It was developed by Bjarne Stroustrup at AT&T's Bell Laboratories.
- Earlier it was called 'C with Classes' but later the name was changed to 'C++'.
- C++ is a superset of C, so that most C programs are also C++ programs.

Header Files in C++

- A header file in C/C++ contains:
  - Function definitions
  - Data type definitions
  - Macros
- Header files offer these features by importing them into your program with the help of a preprocessor directive called **#include.**
- These preprocessor directives are responsible for instructing the C/C++ compiler that these files need to be processed before compilation.

...contd

- **Basically, header files are of 2 types:**
  - **Standard library header files**
    - These are the pre-existing header files already available in the C/C++ compiler.
  - **User-defined header files:**
    - Header files can be designed by the user by saving the file name as .h extension.
- There are a total of 49 header files in the **Standard C++ Library**

Header files in C/C++

- **#include<stdio.h>**
  - **Standard input-output header**
  - Used to perform input and output operations in C like scanf() and printf().
- **#include<string.h>**
  - **String header**
  - Perform string manipulation operations like strlen and strcpy.
- **#include<conio.h>**
  - **Console input-output header**
  - Perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

Header files in C/C++

- **#include<stdlib.h>**
  - **Standard library header**
  - Perform standard utility functions like dynamic memory allocation, using functions such as malloc() and calloc().
- **#include<math.h>**
  - **Math header**
  - Perform mathematical operations like sqrt() and pow().
  - To obtain the square root and the power of a number respectively.
- **#include<ctype.h>**
  - **Character type header**
  - Perform character type functions like isaplha() and isdigit().
  - To find whether the given character is an alphabet or a digit respectively.
- **#include<time.h>**
  - **Time header**
  - Perform functions related to date and time like setdate() and getdate().
  - To modify the system date and get the CPU time respectively.

Header files in C/C++

- **#include<assert.h>**
  - **Assertion header**
  - It is used in program assertion functions like assert().
  - To get an integer data type in C/C++ as a parameter which prints stderr only if the parameter passed is 0.
- **#include<locale.h>**
  - **Localization header**
  - Perform localization functions like setlocale() and localeconv().
  - To set locale and get locale conventions respectively.
- **#include<signal.h>**
  - **Signal header**
  - Perform signal handling functions like signal() and raise().
  - To install signal handler and to raise the signal in the program respectively

Header files in C/C++

- **#include<setjmp.h>**
  - **Jump header**
  - Perform jump functions.
- **#include<stdarg.h>**
  - **Standard argument header**
  - Perform standard argument functions like va_start and va_arg().
  - To indicate start of the variable-length argument list and to fetch the arguments from the variable-length argument list in the program respectively.
- **#include<errno.h>**
  - **Error handling header**
  - Used to perform error handling operations like errno().
  - To indicate errors in the program by initially assigning the value of this function to 0 and then later changing it to indicate errors.

Header Files in C++

- Following are some C++ header files which are not supported in C-
  - **#inlcude<iostream>**
    - **Input Output Stream**
    - Used as a stream of Input and Output.
  - **#include<iomanip.h>**
    - **Input-Output Manipulation**
    - Used to access set() and setprecision().
  - **#include<fstream.h>**
    - **File stream**
    - Used to control the data to read from a file as an input and data to write into the file as an output.

General form of a C++ program

```
// Program description
#include directives
int main()
{
    constant declarations
    variable declarations
    executable statements
    return 0;
}
```

C++ Character Set

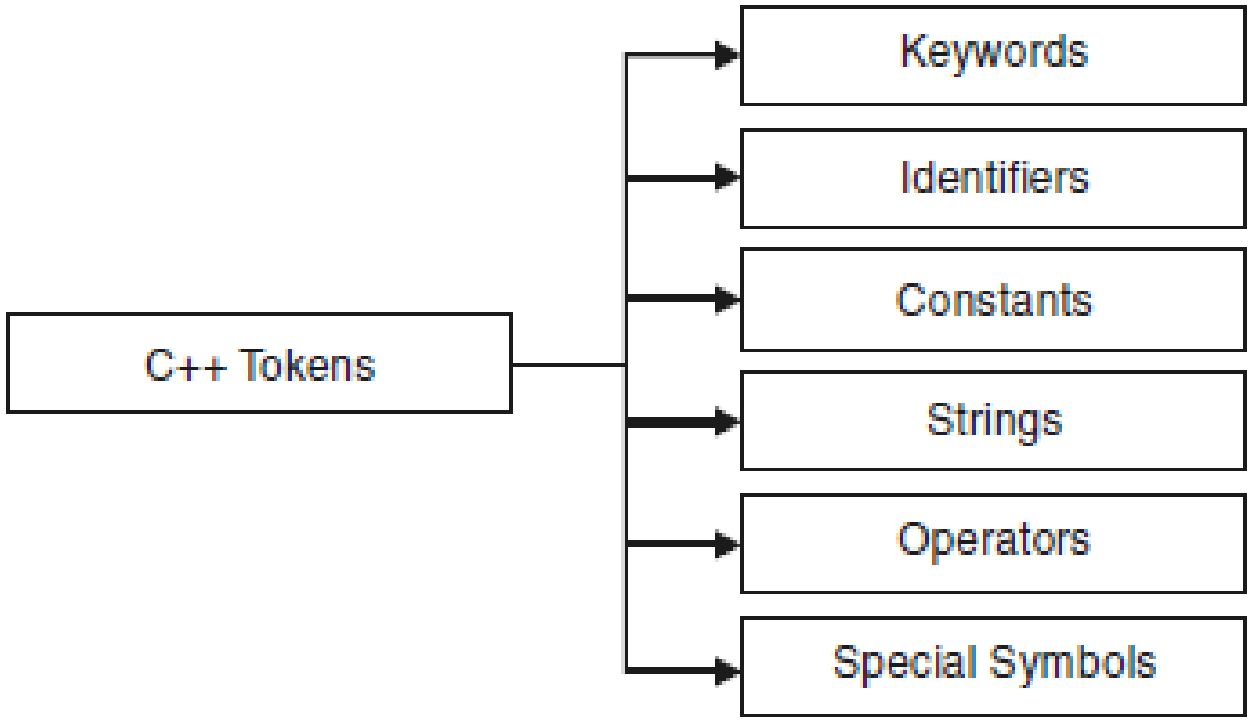
- A C++ program is a collection of number of instructions written in a meaningful order.
- Further, instructions are made up of keywords, variables, functions, objects etc.
- All these uses C++ character set as shown below:

S.No.	*Elements of C++ character set*
1.	Upper Case letters : A to Z
2.	Lower Case letters : a to z
3.	Digits : 0 to 9
4.	Symbols (See below table)

C++ Character Set

Tokens in C++

- Smallest individual unit in a program is called Token.
- C++ defined 6 types of tokens



C++ Keywords

- Each keyword has a predefined purpose in the language.
- Keywords cannot be used as variable and constant names.
- There are 63 keywords in C++ as follows:

asm	auto	bool	break	cae	catch
char	class	const	const_cast	continue	default
delete	do	double	dynamic_cast	else	enum
explicit	export	extern	false	float	for
friend	goto	if	inline	int	long
mutable	namespace	new	operator	private	protected
public	register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch	template
this	throw	true	try	typedef	typeid
typename	union	unsigned	using	virtual	void
volatile	wchar_t	while			

C++ identifiers

- An identifier is a name for a variable, constant, function, etc.
- It consists of a letter followed by any sequence of letters, digits, and underscores.
- Rules for writing identifiers:
  - First letter must be an alphabet or underscore.
  - From second character onwards, any combination of digits, alphabets or underscores are allowed.
  - No other symbol except digits, alphabets or underscores is allowed.
  - Keywords cannot be used as identifiers.
  - Maximum length of identifier depends upon the compiler used.
- Examples:

Valid Identifiers :			
order_no	name	_err	_123
xyz	radius	a23	int_rate
Invalid Identifiers :			
order-no	12name	err	int
x$	s name	hari+45	123

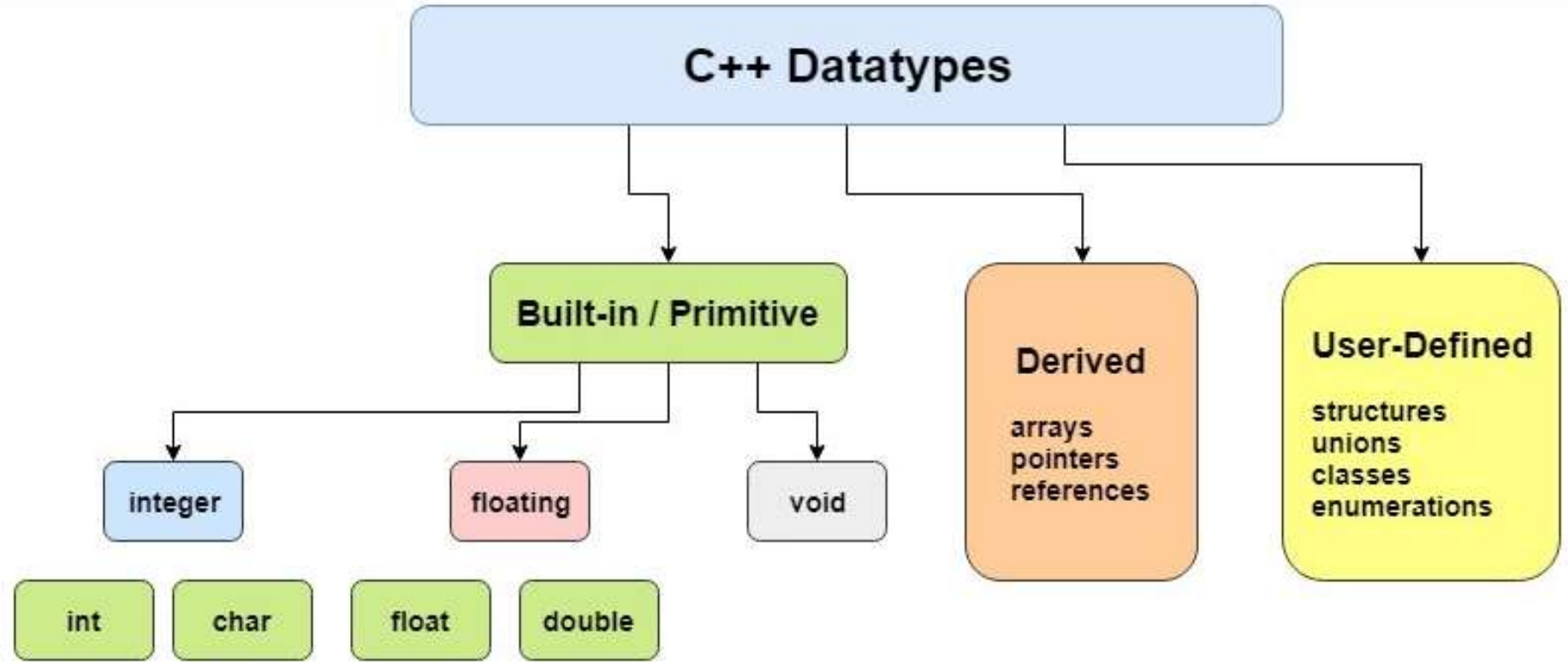
Variables

- It is a named location in memory that is used to hold a value that can be modified in the program by the instruction.
- All the variables must be declared before they can be used.
- General form:

  datatype variable_name [list];

- Unlike C, C++ allows to declare variable anywhere in the program before its use.

Data Types in C++



C++ comments

- Comments are explanatory notes which are ignored by the compiler.
- There are two ways to include comments in a program:

```
// A double slash marks the start of a
//single line comment.

/* A slash followed by an asterisk marks
the start of a multiple line comment. It
ends with an asterisk followed by a
slash. */
```

Programming Style

C++ is a free-format language, which means that:

- Extra blanks (spaces) or tabs before or after identifiers/operators are ignored.
- Blank lines are ignored by the compiler just like comments.
- Code can be indented in any way.
- There can be more than one statement on a single line.
- A single statement can continue over several lines.

A simple C++ program

```
#include <iostream>
using namespace std;
void main()
{
    cout<<"hello C++"<<endl;
}
```

Output Operator

```
cout << "C++ is better than C.";
```

- cout represents the standard output stream.
- The operator << is called **insertion** operator or **put to** operator.

```
#include <iostream.h>
void main()
{
    cout<<123<<endl;
    cout<<23.567<<endl;
    cout<<12345678<<endl;
    cout<<'P'<<endl;
    cout<<"Hari"<<endl;
}
```

Example

```
#include <iostream.h>
void main()
{
        int x(10);
        float f(23.34);
        char ch('P');
        cout<<"x="<<x<<endl;
        cout<<"f="<<f<<endl;
        cout<<"ch="<<ch<<endl;
}
```

OUTPUT :

```
x=10
f=23.34
ch=P
```

Input Operator

```
cin >> number1;
```

- cin represents the standard input stream.
- The operator >> is called **extraction** operator or **get from** operator.

```
cin>>var1>>var2>>var2.....>>var n
```

Example Program

```
#include <iostream.h>
void main()
{
    int x;
    cout<<"Enter a number"<<endl;
    cin>>x;
    cout<<"You have entered"<<endl;
    cout<<"x :="<<x<<endl;
}
```

OUTPUT :

```
Enter a number
40
You have entered
x :=40
```

Program to add two numbers

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

int main()
{
    int firstNumber, secondNumber, sumOfTwoNumbers;

    cout << "Enter two integers: ";
    cin >> firstNumber >> secondNumber;

    // sum of two numbers in stored in variable sumOfTwoNumbers
    sumOfTwoNumbers = firstNumber + secondNumber;

    // Prints sum
    cout << firstNumber << " + " <<  secondNumber << " = " << sumOfTwoNumbers;

    return 0;
}
```

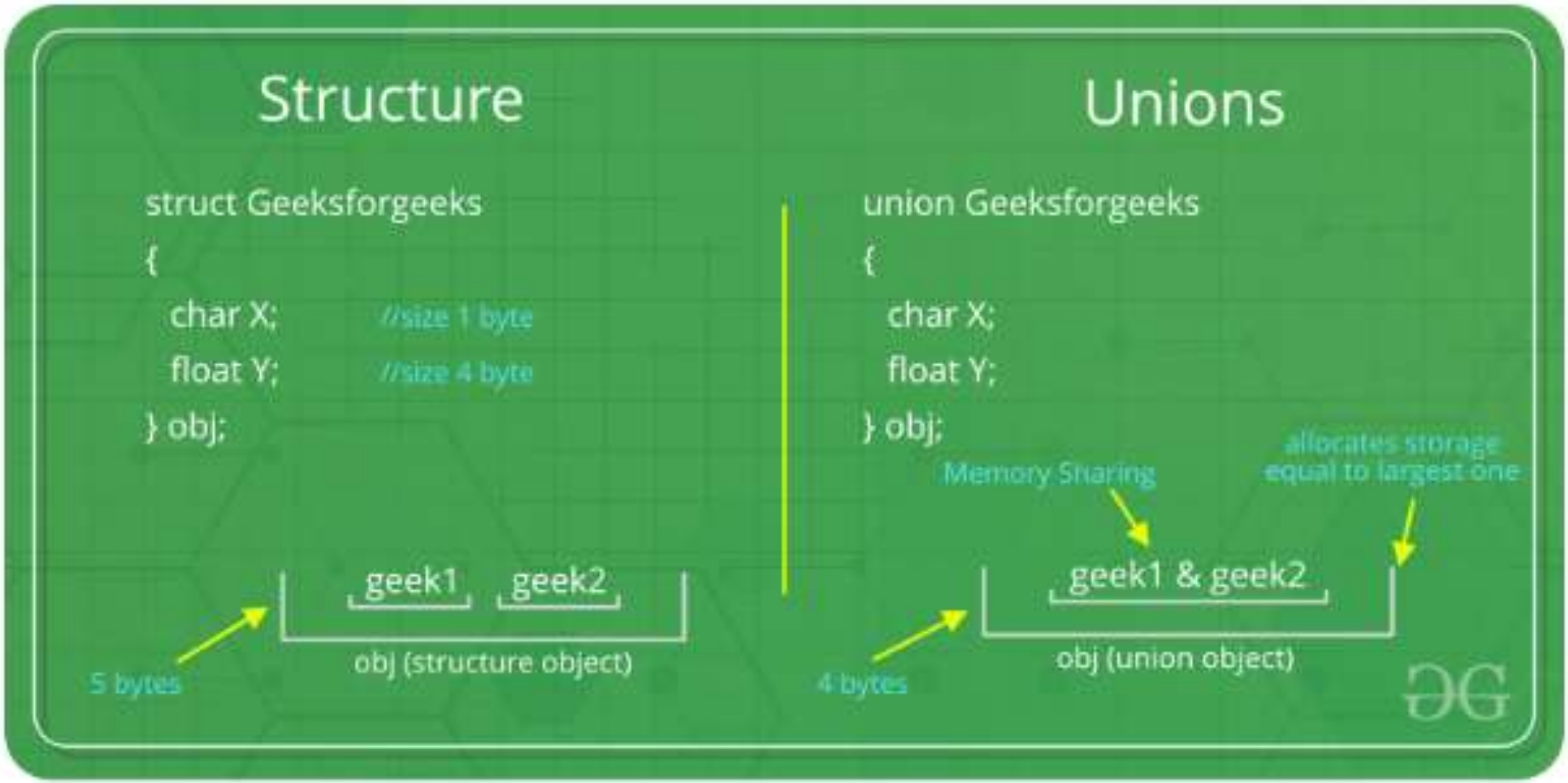
Output

```
Enter two integers: 4
5
4 + 5 = 9
```

Structure & Union

- Structure and Union both are a collection of heterogeneous data elements.

```
struct book
{
    char title[25];
    char author[25];
    int pages;
    float price;
} book1, book2,book3;

union result
{
    int marks;
    char grade;
    float percent;
};
```


Structure
struct Geeksforgeeks
{
char X; //size 1 byte
float Y; //size 4 byte
} obj;
geek1
geek2
obj (structure object)
5 bytes
Unions
union Geeksforgeeks
{
char X;
float Y;
} obj;
Memory Sharing
allocates storage
equal to largest one
geek1 & geek2
obj (union object)
4 bytes

Difference between Structure & Union

Structure

- A structure is defined with struct keyword.
- All members of structure can be manipulated simultaneously.
- Size of structure object is equal to the sum of individual sizes of member objects.
- Structure members are allocated distinct memory locations.
- Structures are not memory efficient as compared to unions.

Union

- A union is defined with union keyword.
- The members of a union can be manipulated one at a time.
- The size of a union object is equal to the size of largest member object.
- Union members share common memory space for their exclusive usage.
- Unions are considered memory efficient where members are not required to be accessed simultaneously.

Store and Display information using Structure

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

struct student
{
    char name[50];
    int roll;
    float marks;
};

int main()
{
    student s;
    cout << "Enter information," << endl;
    cout << "Enter name: ";
    cin >> s.name;
    cout << "Enter roll number: ";
    cin >> s.roll;
    cout << "Enter marks: ";
    cin >> s.marks;

    cout << "\nDisplaying Information," << endl;
    cout << "Name: " << s.name << endl;
    cout << "Roll: " << s.roll << endl;
    cout << "Marks: " << s.marks << endl;
    return 0;
}
```

Output

```
Enter information,
Enter name: Bill
Enter roll number: 4
Enter marks: 55.6

Displaying Information,
Name: Bill
Roll: 4
Marks: 55.6
```

```
#include<iostream.h>

union Employee
{
      int Id;
      char Name[25];
      int Age;
      long Salary;
};

void main()
{

      Employee E;

            cout << "\nEnter Employee Id : ";
            cin >> E.Id;

            cout << "\nEnter Employee Name : ";
            cin >> E.Name;

            cout << "\nEnter Employee Age : ";
            cin >> E.Age;

            cout << "\nEnter Employee Salary : ";
            cin >> E.Salary;

            cout << "\n\nEmployee Id : " << E.Id;
            cout << "\nEmployee Name : " << E.Name;
            cout << "\nEmployee Age : " << E.Age;
            cout << "\nEmployee Salary : " << E.Salary;
```

Output :

```
Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000

Employee Id : -20536
Employee Name : ?$?$  □□?
Employee Age : -20536
Employee Salary : 45000
```

```
#include<iostream.h>

union Employee
{
      int Id;
      char Name[25];
      int Age;
      long Salary;
};

void main()
{

      Employee E;

            cout << "\nEnter Employee Id : ";
            cin >> E.Id;
            cout << "Employee Id : " << E.Id;

            cout << "\n\nEnter Employee Name : ";
            cin >> E.Name;
            cout << "Employee Name : " << E.Name;

            cout << "\n\nEnter Employee Age : ";
            cin >> E.Age;
            cout << "Employee Age : " << E.Age;

            cout << "\n\nEnter Employee Salary : ";
            cin >> E.Salary;
            cout << "Employee Salary : " << E.Salary;

}
```

Output :

```
Enter Employee Id : 1
Employee Id : 1

Enter Employee Name : Kumar
Employee Name : Kumar

Enter Employee Age : 29
Employee Age : 29

Enter Employee Salary : 45000
Employee Salary : 45000
```

Enumerated data type

- User defined data type which provide a way of attaching names to numbers.
- enum automatically enumerates a list of words by assigning them values 0,1,2,….

```
enum shape{circle, square, triangle};
enum color{red, blue, green, yellow};
enum position{on, off};
```

- We can use these as typenames also like:

```
shape ellipse;
color background;
```

- The integers values assigned to enumerators can also be over-ride. For example:

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

```c
// An example program to demonstrate working
// of enum in C
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

Output

2

```c
// Another example program to demonstrate working
// of enum in C
#include<stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,
          Aug, Sep, Oct, Nov, Dec};

int main()
{
    int i;
    for (i=Jan; i<=Dec; i++)
        printf("%d ", i);

    return 0;
}
```

Output:

```
0 1 2 3 4 5 6 7 8 9 10 11
```

```c
#include <stdio.h>
enum day {sunday = 1, monday, tuesday = 5,
          wednesday, thursday = 10, friday, saturday};

int main()
{
    printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,
           wednesday, thursday, friday, saturday);
    return 0;
}
```

Output:

```
1 2 5 6 10 11 12
```

Derived data types

- Arrays
- Functions
- Pointers
- Reference

Pointers

```
int *ip;        //int pointer
ip=&x;          //address of x assigned to ip
*ip=10;         //10 assigned to x through indirection

char * const ptr1='Good'; //constant pointer
```

- Address of ptr1 cannot be modified.

Reference variables

- Reference variable provides as alias name for a previously defined variable.
- Syntax:

  Data-type & reference-name = variable-name
- eg.

  float total=100;

  float & sum = total;
- **sum** is the alternative name declared to represent t**otal**.
- Any modification or assignment applied to either variable will effect both.
- Reference variable must be initialized at the time of declaration.

Memory Management Operators

- New
  - To allocate memory dynamically.
- Delete
  - To deallocate memory as and when required.
- An object created using **new** will remain in existence until it is explicitly destroyed by **delete**.

Using NEW operator

- General form of using new operator:

  Pointer-variable = new data-type;

- Eg.

  P = new int; //p is a pointer of type int

  Q = new float; //q is a pointer of type float

  - Here, p and q must be already declared.

- Alternatively,

  int *p = new int;

  float *q = new float;

  *p = 25;

  *q = 7.5;

  - Assigns 25 to newly created int object and 7.5 to float object.

  OR

- Pointer-variable = new data-type(value);
  - int *P = new int(25);
  - float *q = new float(7.5);

Using DELETE operator

- delete pointer-variable;

  delete p;

  delete q;

Manipulators

- Endl
  - causes a linefeed to be inserted as new line character (“\n”)
- Setw
  - Specifies a field width for printing the value of a variable and make it right justified.

```
#include <iostream>
#include <iomanip>    // for setw

using namespace std;

int main()
{
     int Basic = 950, Allowance = 95, Total = 1045;

     cout  << setw(10) << "Basic" << setw(10) << Basic << endl
           << setw(10) << "Allowance" << setw(10) << Allowance << endl
           << setw(10) << "Total" << setw(10) << Total << endl;

     return 0;
}
```

```
     Basic     950
 Allowance      95
     Total    1045
```

using namespace std

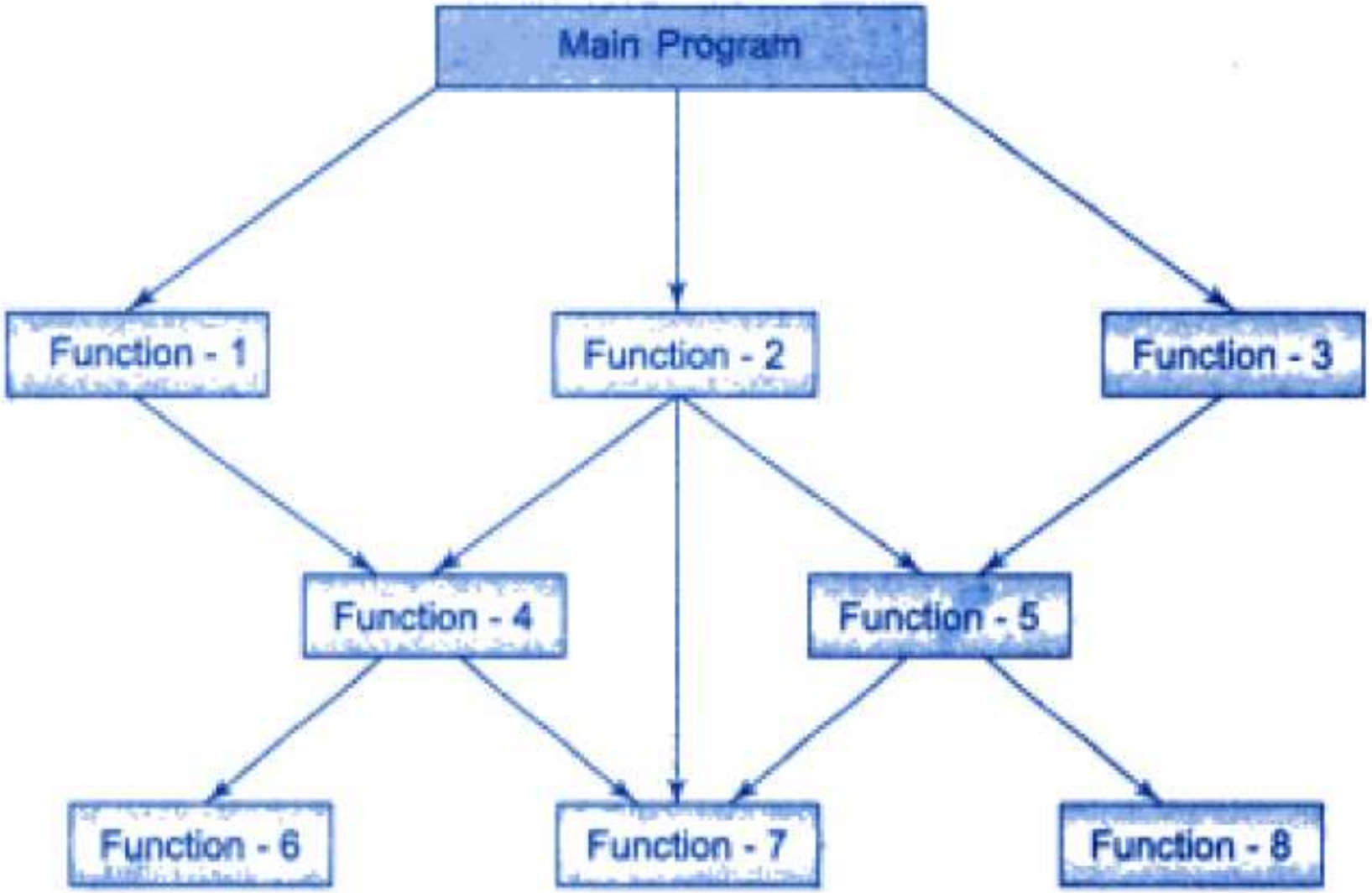
- **“using namespace std”**
  - means we **use** the **namespace** named **std**.
  - **“std”** is an abbreviation for standard which means we **use** all the things with in **“std” namespace**.

Features of Object Oriented Programming in C++

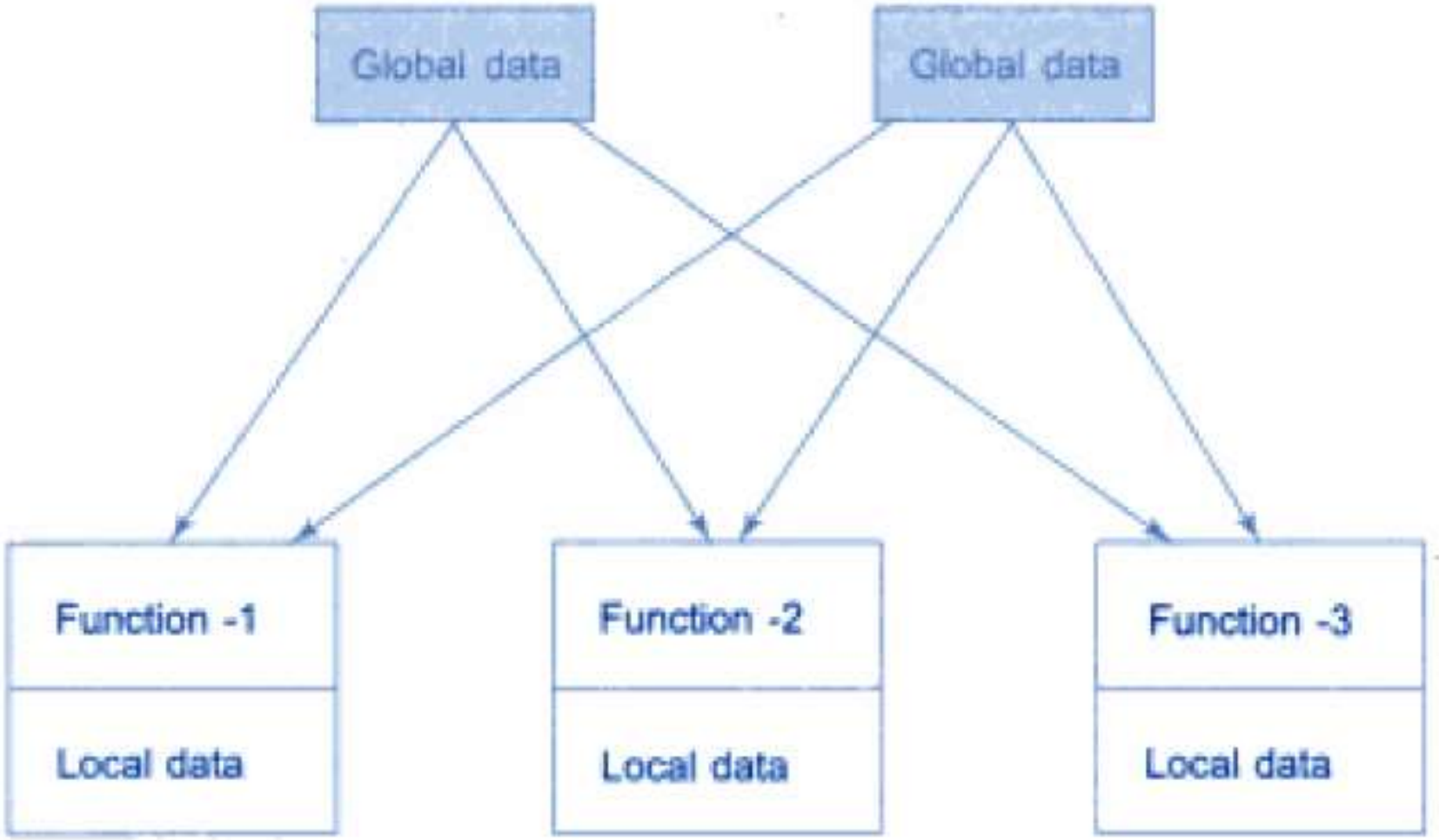
Procedure Oriented Programming

- Conventional programming such as FORTRAN, COBOL, C use procedure oriented programming.
- The number of functions are written to accomplish various tasks.
- Employs top down approach.
- Large programs are divided in small programs known as functions.
- Most functions share global data & data moves freely from one function to another.

Structure of Procedure Oriented Programs



Relationship of Data and Functions in Procedural Programming



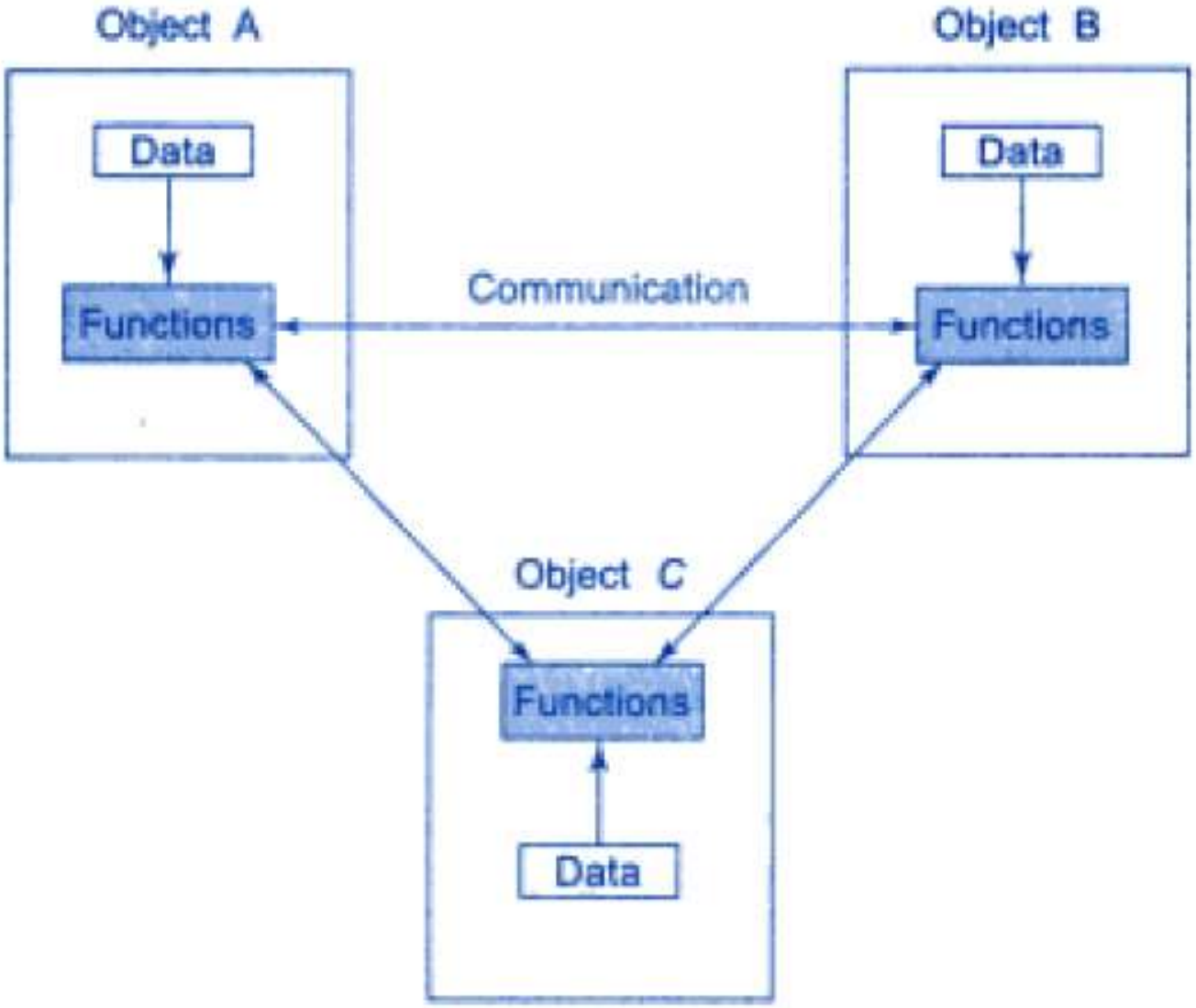
OBJECT ORIENTED PROGRAMMING

- The object oriented approach is used to remove the flaws encountered in procedure oriented approach.
- OOP treats data as a critical element & doesn't allow it to flow freely around the system.
- It ties data more closely to the functions that operate on it.
- OOP allows decomposition of a problem into a number of entities known as **objects** and then builds data and functions around these objects.
- The data of an object can be accessed only by the functions associated with that object, however, functions of one object can access the functions of other objects.

Features of OOP

- Emphasis is on data rather than procedure.
- Programs are decomposed into various entities known as objects.
- Functions that operate on data of an object are tied together.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions may be added easily as per necessity.
- Follows bottom up approach in program design.

Organization of data and functions in OOP



Applications of OOP

- Real Time Systems.
- Simulation and Modeling.
- Object oriented databases.
- AI and Expert System.
- Neural Networks and parallel programming.
- Decision support and office automation systems etc.

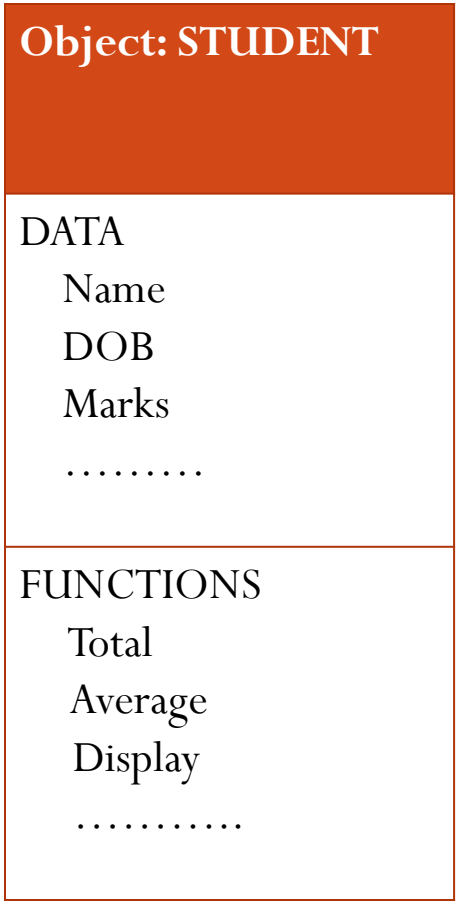
FEATURES OF OOP

- Various features of object oriented programming are:
  - Objects
  - Classes
  - Data Abstraction and Encapsulation
  - Inheritance
  - Polymorphism
  - Dynamic Binding
  - Message Passing

OBJECTS

- Objects are the basic run time entities in an object oriented system.
- They may represent a person, a place, a bank account or any other item that the program has to handle.
- An object takes space in memory and have an associated address like a structure in C.
- During execution, objects interacts by sending messages to each other.
- Each object contains data and the code to manipulate the data.

REPRESENTING AN OBJECT



CLASSES

- A class is a collection of objects of similar types

  e.g. mango, apple are objects of class Fruit

- Classes are user defined data types and behaves like built-in types.

- Any number of objects can be created after defining a class.
- Syntax:

  fruit mango;

- **Syntax for creating an object is similar to the syntax used in C to declare an integer object.**

  e.g. int num;

ENCAPSULATION

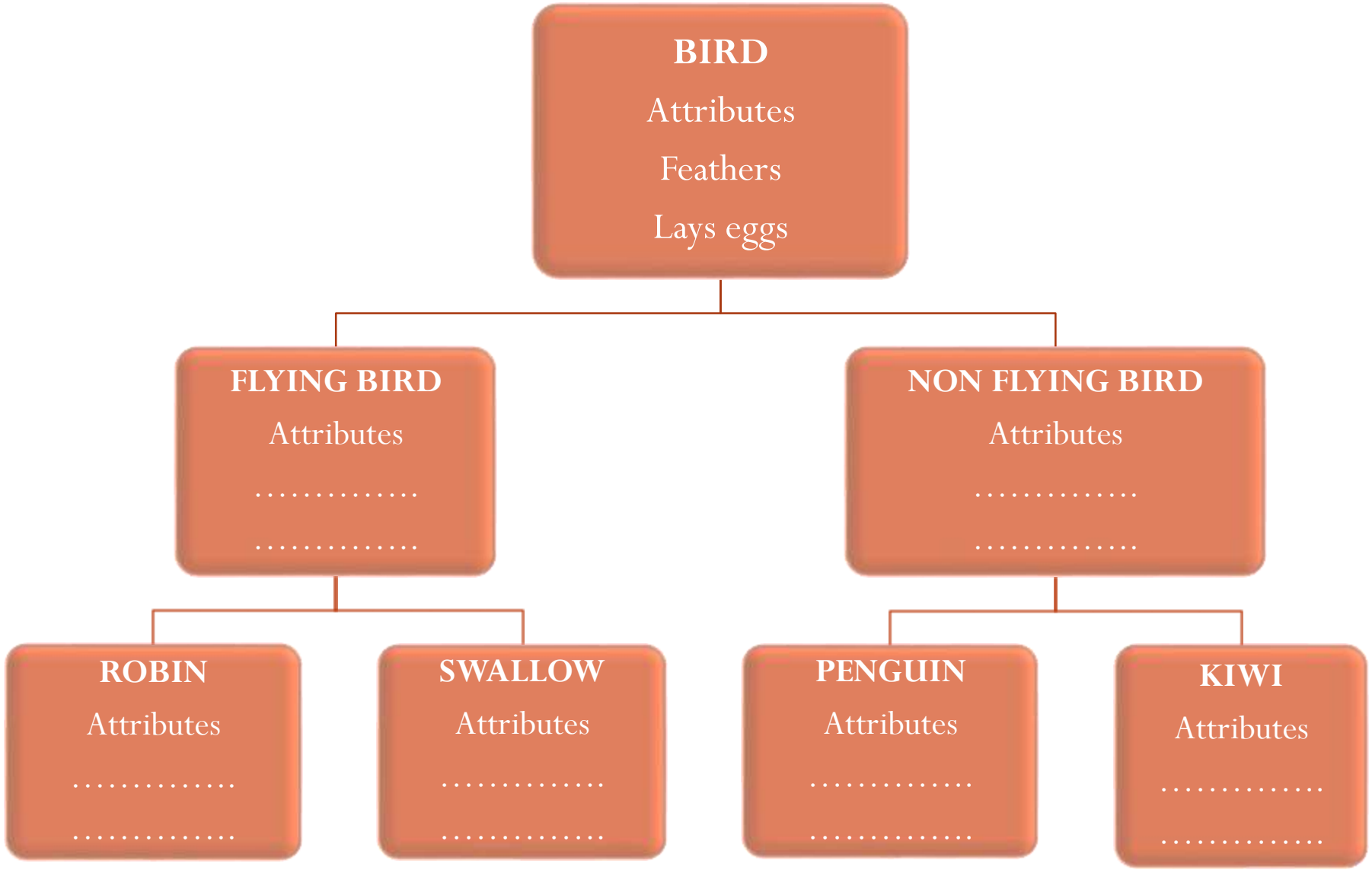
- The wrapping up of data and functions into a single unit is called **Encapsulation**.
- Data is not accessible to the outside world.
- Only the functions that are wrapped in the class can access it.
- It is called **Data Hiding** or **Information Hiding**

DATA ABSTRACTION

- Representing essential details without including background details is called **Abstraction**.
- Classes are also known as **Abstract Data Types** (ADT).
- The variables or attributes are called **Data Members** because they hold information.
- The functions that operate on these data are called **Methods or Member Functions**.

INHERITANCE

- The property by which objects of one class acquire the properties of objects of another class.
- It supports **Reusability**.
- A new class can be derived from the existing class by adding additional features to the existing class without modifying it.
- The new class will have combined features of both the classes.


BIRD
Attributes
Feathers
Lays eggs
FLYING BIRD
Attributes
.....
.....
NON FLYING BIRD
Attributes
.....
.....
ROBIN
Attributes
.....
.....
SWALLOW
Attributes
.....
.....
PENGUIN
Attributes
.....
.....
KIWI
Attributes
.....
.....

Types of Inheritance

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

POLYMORPHISM

- Ability to take more than one form.
- An operation may exhibit different behavior in different instances.
- The behavior depends upon the types of data used in the operation.

- Types of Polymorphism:
  - **OPERATOR OVERLOADING**
  - **FUNCTION OVERLOADING**

OPERATOR OVERLOADING

- Process of making an operator to exhibit different behavior in different instances is called **Operator Overloading**.

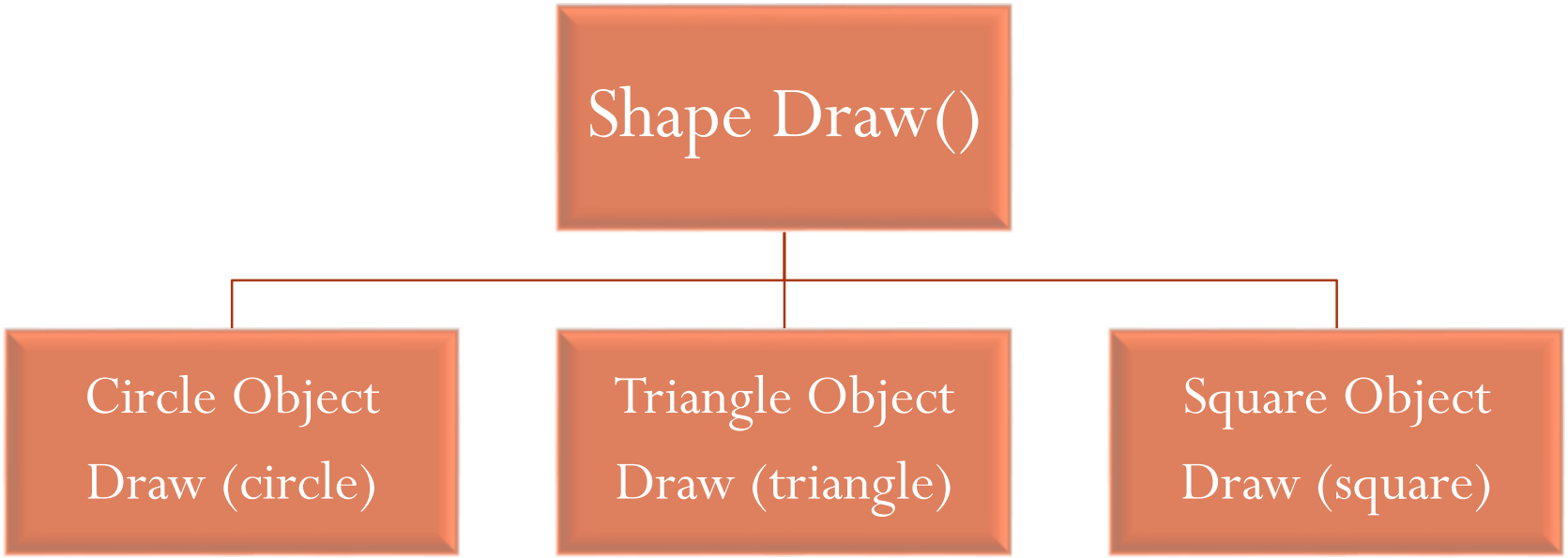
- E.g. Consider the addition operation:
  - for 2 numbers, the operation will generate a sum
  - If the operands are strings, the operation will produce a third string by concatenation.

Function Overloading

- Using a single function name to perform different types of tasks is called **Function Overloading**.
- Known as **Function Polymorphism.**

- All the functions having same name contains different argument lists.

- Correct function to be invoked depends upon the number and type of arguments.


Shape Draw()
Circle Object
Draw (circle)
Triangle Object
Draw (triangle)
Square Object
Draw (square)

```
// Function volume() is overloaded three times

#include <iostream>

using namespace std;

// Declarations (prototypes)
int volume(int);
double volume(double, int);
long volume(long, int, int);

int main()
{
        cout << volume(10) << "\n";
        cout << volume(2.5,8) << "\n";
        cout << volume(100L,75,15) << "\n";

        return 0;
}

// Function definitions

int volume(int s)  // cube
{
        return(s*s*s);
}

double volume(double r, int h)   // cylinder
{
        return(3.14519*r*r*h);
}

long volume(long l, int b, int h) // rectangular box
{
        return(l*b*h);
}
```

Dynamic Binding

- Also called **Late Binding**.
- It means that the code associated with a given procedure call is not known until the time of call at run time.
- Associated with Polymorphism and Inheritance

MESSAGE PASSING

- Objects in Object Oriented Programming communicate with each other by sending and receiving information to and from other objects.

- A message for an object is a request for the execution of a procedure.

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

  E.g. employee.salary(name);

Program to find largest of three numbers

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

int main()
{
    float n1, n2, n3;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    if(n1 >= n2 && n1 >= n3)
    {
        cout << "Largest number: " << n1;
    }

    if(n2 >= n1 && n2 >= n3)
    {
        cout << "Largest number: " << n2;
    }

    if(n3 >= n1 && n3 >= n2) {
        cout << "Largest number: " << n3;
    }

    return 0;
}
```

Output

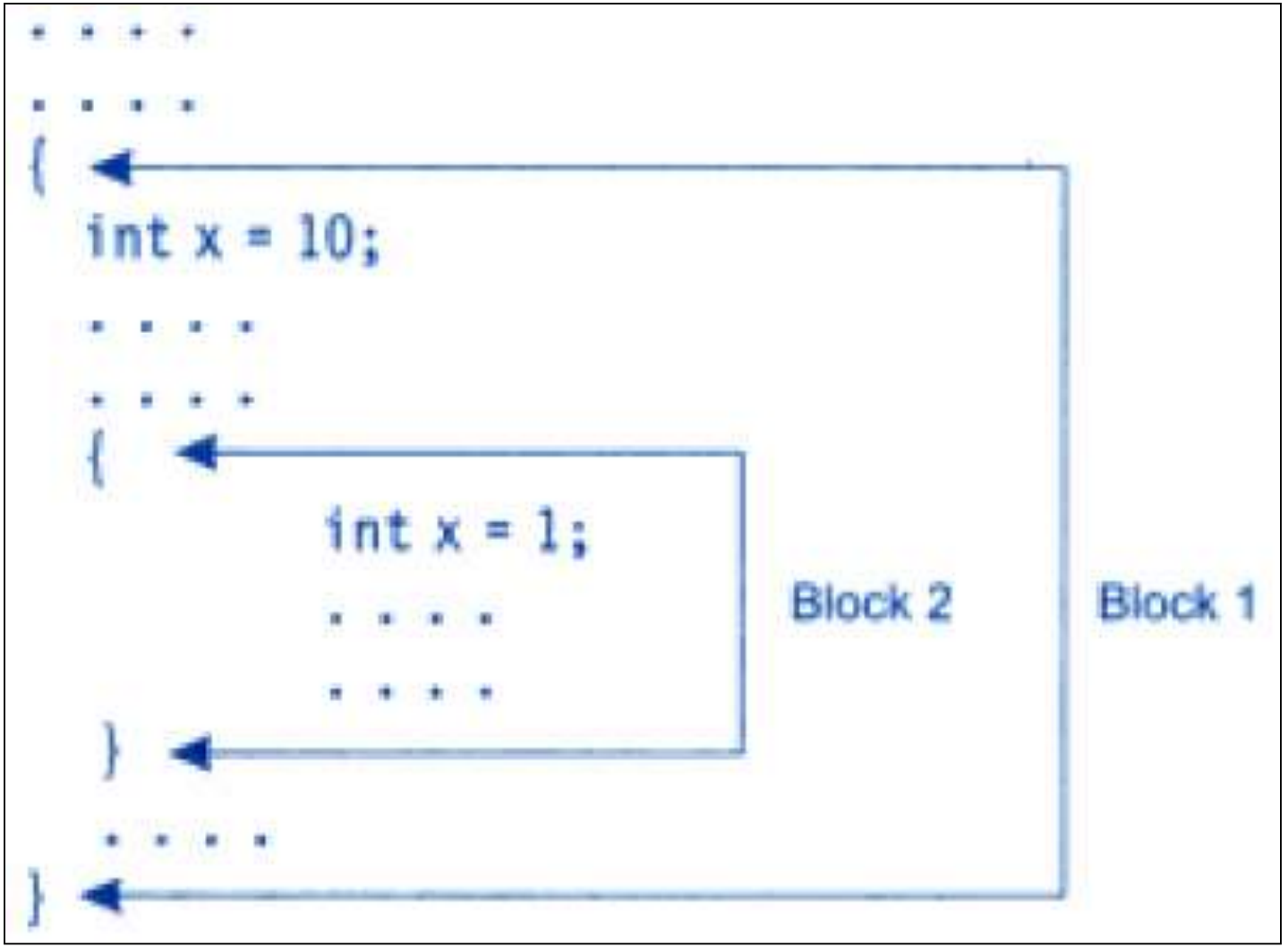
```
Enter three numbers: 2.3
8.3
-4.2
Largest number: 8.3
```

Scope Resolution Operator

- The operator :: is called scope resolution operator.
- It is used for two purposes.
  - For accessing global variables
  - Identifying class members to which class they belong

- The scope of a variable extends from the point of its declaration till the end of the block containing the declaration.
- The variable is local to that block.
- If a global variable also exists with same name as the local variable, then local variable will be called by default within that block.
- Scope resolution operator helps to access the global variable inside the block.
- Syntax: **:: variable-name**

Scope Resolution Operator



SCOPE RESOLUTION OPERATOR

```
#include <iostream>

using namespace std;
int m = 10;              // global m

int main()
{
        int m = 20;    // m redeclared, local to main

        {
                int k = m;
                int m = 30;    // m declared again
                               // local to inner block

                cout <<  "we are in inner block \n";
                cout <<  "k = " << k << "\n";
                cout <<  "m = " << m << "\n";
                cout <<  "::m = " << ::m << "\n";
        }

        cout << "\nWe are in outer block \n";
        cout << "m = " << m << "\n";
        cout << "::m = " << ::m << "\n";

        return 0;
}
```

OUTPUT

```
We are in inner block
k = 20
m = 30
::m = 10

We are in outer block
m = 20
::m = 10
```

Functions in C++

- A function is a block of code that performs a specific task.
- There are two types of functions:
  - Standard library functions (**Predefined in C++**)
  - User-defined function (**Created by users**)

- Basic Syntax:

```
return-type function-name(parameter1, parameter2, ...)
{
    // function-body
}
```

  - **return-type:** suggests what function will return
  - **function-name:** name of the function
  - **Parameters:** variables to hold values of arguments passed while function is called. A function may or may not contain parameters.
  - **Function body:** part where the code statements are written

Declaring, Defining and Calling a function

- A function **declaration** tells the compiler about a function's name, return type, and parameters.
- Syntax:

```
return_type function_name( parameter list );
```

- A function **definition** provides the actual body of the function.
- Syntax:

```
return_type function_name( parameter list ) {
   body of the function
}
```

- To use a function, we have to **call or invoke** that function.
- When a program calls a function, program control is transferred to the called function.
- A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the calling program.

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

// function declaration
int max(int num1, int num2);

int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;
   int ret;

   // calling a function to get max value.
   ret = max(a, b);
   cout << "Max value is : " << ret << endl;

   return 0;
}

// function returning the max between two numbers
int max(int num1, int num2) {
   // local variable declaration
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Output

```
Max value is : 200
```

Inline Functions

- When a function is called it performs a series of instructions:
  - Jumping to the function
  - Saving registers
  - Pushing arguments into stack
  - Returning to calling function
- This might be an overhead if function is small.
- **Inline functions** can to be used as a solution for this.

Inline Functions

- It is a function that is expanded in line when it is invoked.
- The compiler replaces the function call with corresponding function code.
- Syntax:

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

- Eg.

```
inline double cube(double a)
{
        return (a*a*a);
}
```

- It can by invoked by the statement like:

```
        c=cube(3.0);
```

- All inline functions must be defined before they are called.

- There are certain situations where inline expansion may not work:
  - For function returning values, if a loop, switch or goto exists.
  - For function not returning values, if a return statement exists.
  - If function contain static variables.
  - If inline functions are recursive.

Program: Inline Function

```
#include <iostream>

using namespace std;

        inline float mul(float x, float y)
        {
                return(x*y);
        }

        inline double div(double p, double q)
        {
                return(p/q);
        }

    int main()
    {
        float a = 12.345;
        float b = 9.82;

        cout << mul(a,b) << "\n";
        cout << div(a,b) << "\n";

        return 0;
    }
```

Default Arguments

- C++ allows to call a function without specifying all its arguments.
- The function assigns a default value to the parameter which does not have a matching argument in the function call.
- Default values are specified when the function is declared.

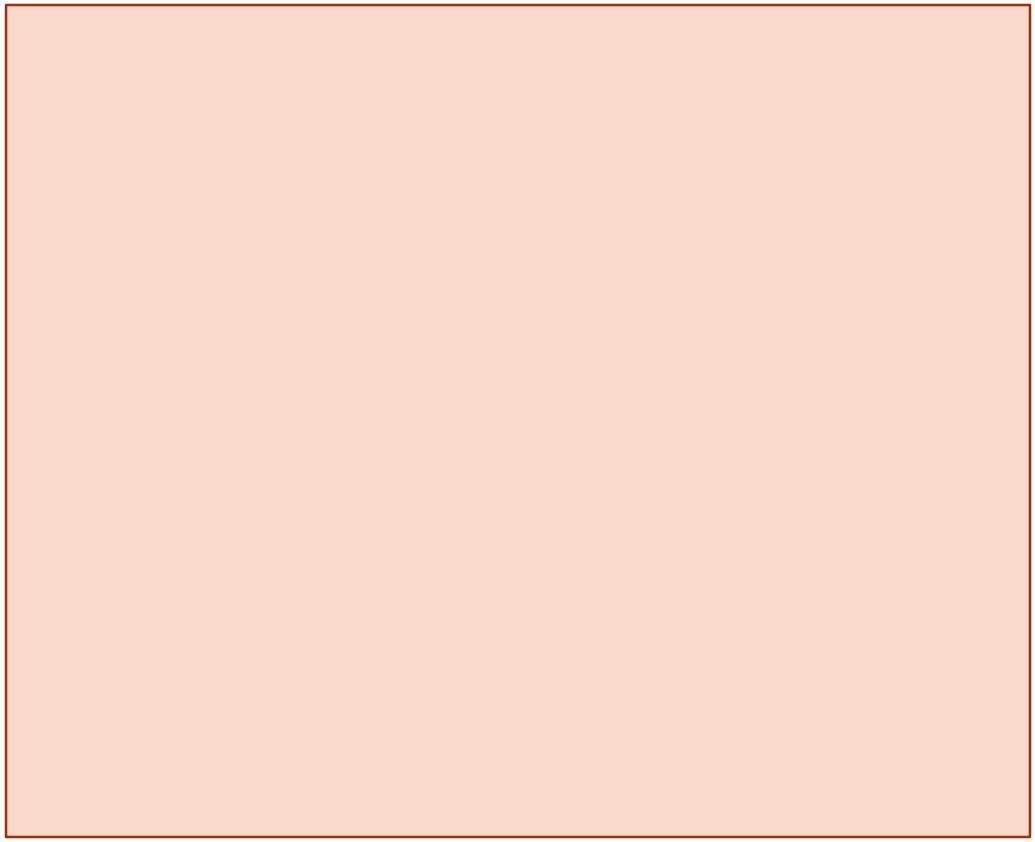
```cpp
#include <iostream>
using namespace std;

int sum(int a, int b = 20) {
   int result;
   result = a + b;

   return (result);
}
int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;
   int result;

   // calling a function to add the values.
   result = sum(a, b);
   cout << "Total value is :" << result << endl;

   // calling a function again as follows.
   result = sum(a);
   cout << "Total value is :" << result << endl;

   return 0;
```

Output

```
Total value is :300
Total value is :120
```

Classes

- A class is a way to bind the data and its associated functions together.
- General form of class declaration:

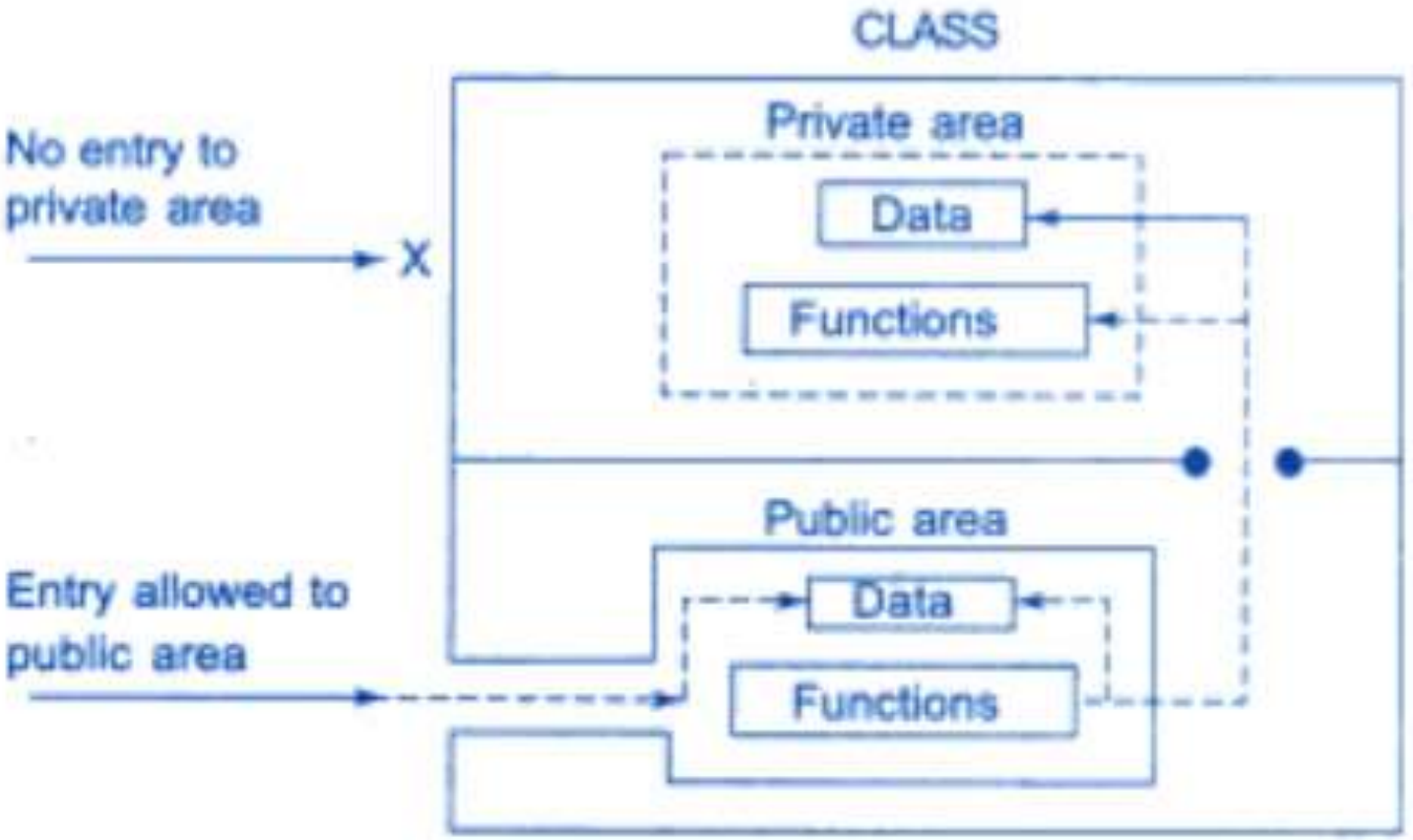


Visibility Labels

- Private
  - Can be accessed from within the class.
  - Use of keyword is optional.
  - Class members are by default private.
- Public
  - Can be accessed from outside the class also.

- If both labels are missing, by default, all the members of the class are private.

Data hiding in classes



Accessing class members

- General form:

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

- Eg.

  x.getdata(100,75.5);

  x.putdata();

```
class xyz
{
      int x;
      int y;
  public:
      int z;
};
      .....
      .....
xyz p;
p.x = 0;          // error, x is private
p.z = 10          // OK, z is public
      .....
      .....
```

Defining member functions

- Member functions can be defined in two places:
  - Outside the class definition
  - Inside the class definition

```
#include <iostream>

using namespace std;

class item
{
        int number;     // private by default
        float cost;     // private by default
    public:
        void getdata(int a, float b);       // prototype declaration,
                                            // to be defined

        // Function defined inside class
        void putdata(void)
        {
               cout << "number :" << number << "\n";
               cout << "cost   :" << cost   << "\n";
        }
};
//............. Member Function Definition .................
void item :: getdata(int a, float b)      // use membership label
{
        number = a;     // private variables
        cost = b;       // directly used
}
//...................... Main Program ......................

int main()
{
        item x;// create object x

        cout <<  "\nobject x "  <<  "\n";

        x.getdata(100, 299.95);             // call member function
        x.putdata();                        // call member function

        item y;                             // create another object

        cout <<  "\nobject y"  << "\n";

        y.getdata(200, 175.50);
        y.putdata();

        return 0;
}
```

Nesting of Member Functions

```
#include <iostream>

using namespace std;

class set
{
        int m, n;
  public:
        void input(void);
        void display(void);
        int largest(void);
};

int set :: largest(void)
{
        if(m >= n)
                return(m);
        else
                return(n);
}
```

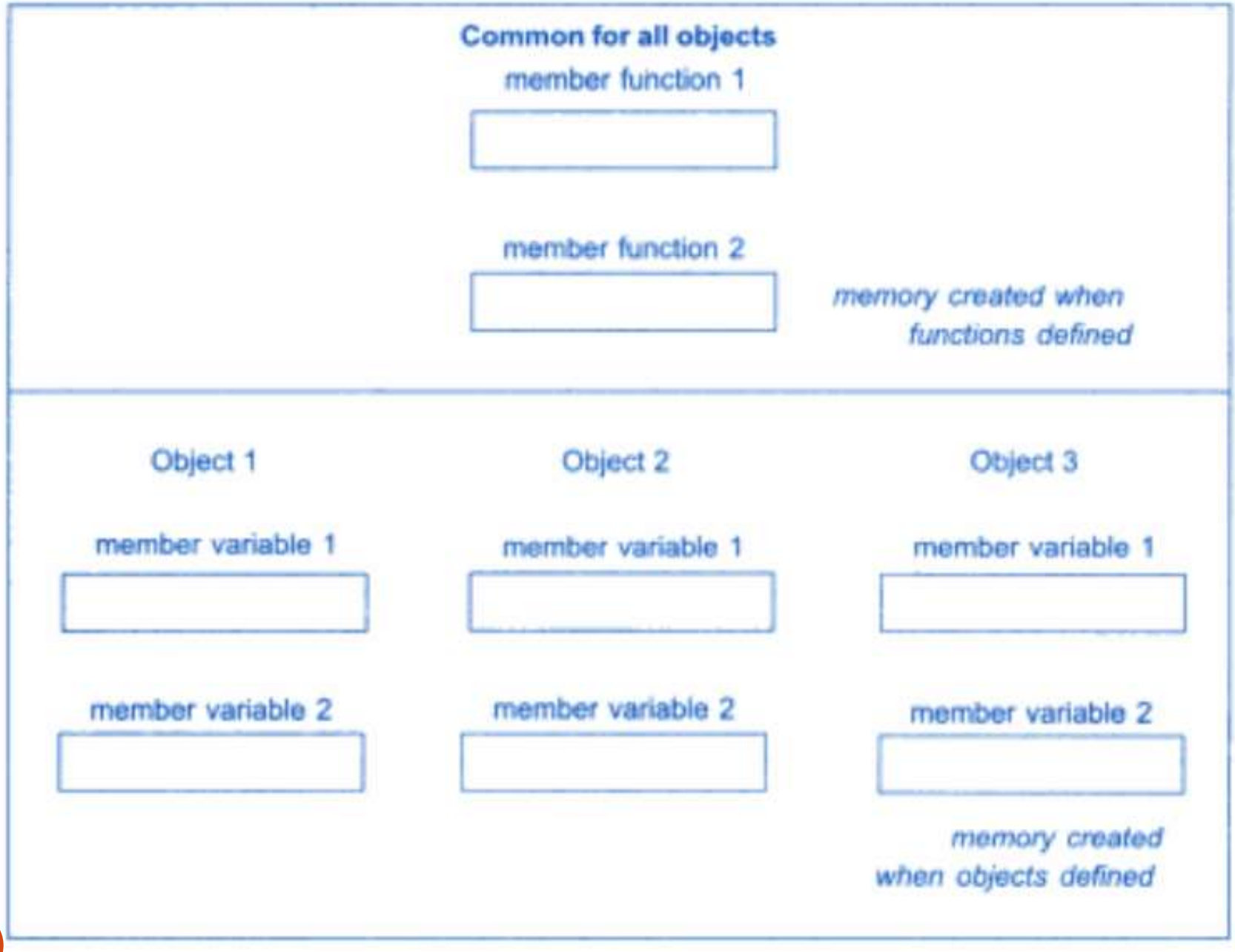
```
void set :: input(void)
{
        cout << "Input values of m and n" << "\n";
        cin >> m >> n;
}

void set :: display(void)
{
        cout << "Largest value = "
             << largest() << "\n";
}

int main()
{
        set A;
        A.input();
        A.display();

        return 0;
}
```

Private Member Functions

- A private member function can only be called by another function that is a member of its class.
- An object of the class cannot call the private member function.

Memory allocation for objects

- Memory space to members of class is allocated as:
  - Member functions
    - When they are defined as a part of class specification.
    - All the objects of that class use the same member functions.
  - Data members
    - When the objects of the class are created.
    - Separate space is allocated to each data member for each object.


Common for all objects
member function 1
member function 2
memory created when functions defined
Object 1
Object 2
Object 3
member variable 1
member variable 1
member variable 1
member variable 2
member variable 2
member variable 2
memory created when objects defined

Static Data Members

- A static data member has following characteristics:
  - Initialized to zero when first object of the class is created.
  - Cannot be initialized in any other way.
  - Only one copy of the static member is created and shared by all objects of the class.
  - Its lifetime is the entire program.

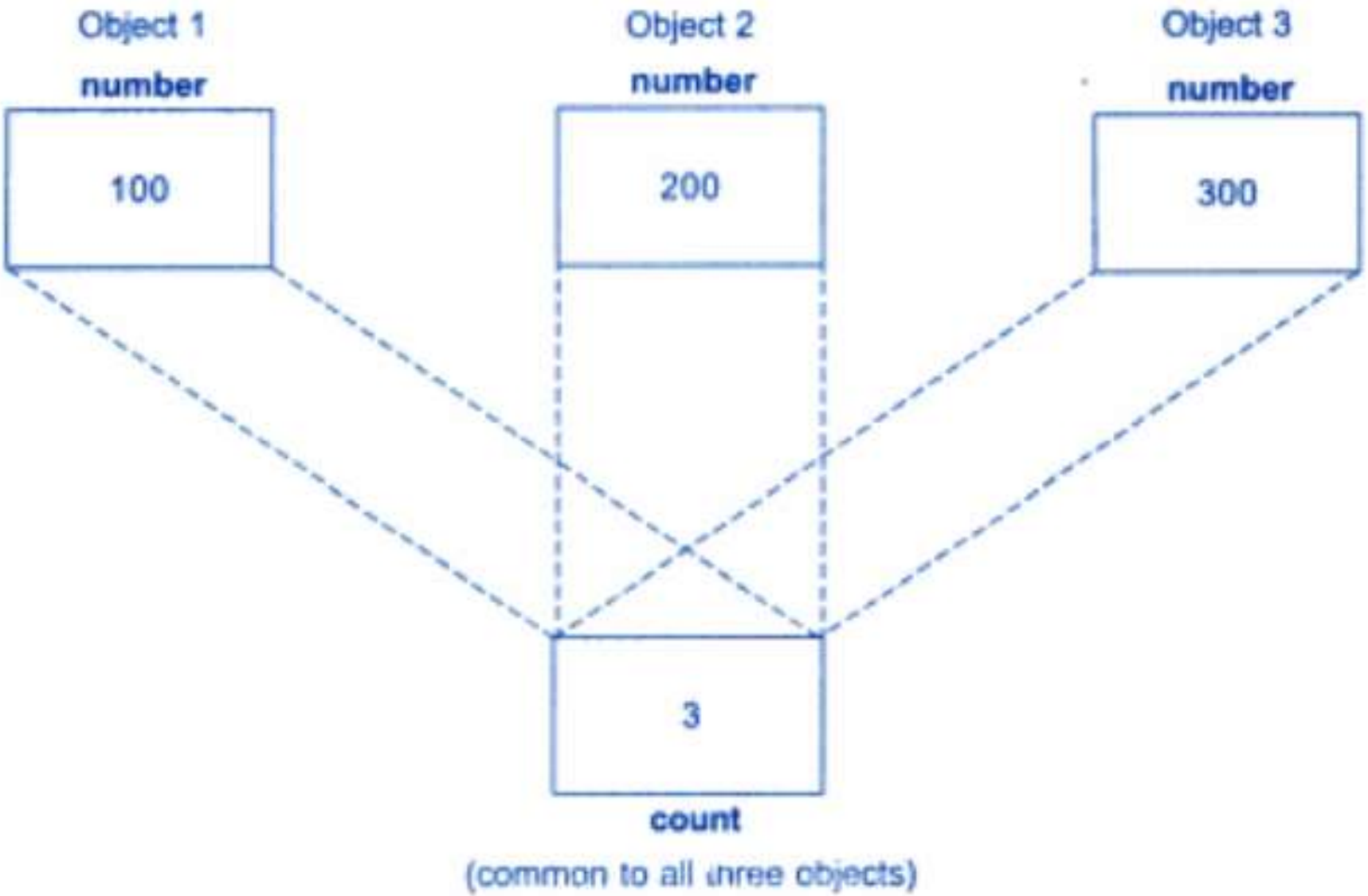
STATIC CLASS MEMBER

```
#include <iostream>

using namespace std;

class item
{
        static int count;
        int number;
  public:
        void getdata(int a)
        {
                number = a;
                count ++;
        }
        void getcount(void)
        {
                cout << "count: ";
                cout << count  << "\n";
        }
};
```

```
int item :: count;

int main()
{
    item a, b, c;           // count is initialized to zero
    a.getcount();           // display count
    b.getcount();
    c.getcount();

    a.getdata(100);         // getting data into object a
    b.getdata(200);         // getting data into object b
    c.getdata(300);         // getting data into object c

    cout << "After reading data" << "\n";

    a.getcount();           // display count
    b.getcount();
    c.getcount();
    return 0;
}
```

Output

```
count: 0
count: 0
count: 0
After reading data
count: 3
count: 3
count: 3
```


Object 1
number
100
Object 2
number
200
Object 3
number
300
3
count
(common to all three objects)

Static Member Functions

- A static member function can have access to only static data members declared in the same class.
- Static member function can be called using the class name as:

  class-name :: function-name;

```
#include <iostream>

using namespace std;

class test
{
        int code;
        static int count;              // static member variable
public:
        void setcode(void)
        {
        code = ++count;
        }
        void showcode(void)
        {
        cout << "object number: " << code << "\n";
        }
        static void showcount(void)    // static member function
        {
        cout << "count: " << count << "\n";
        }
};
```

```
int test :: count;
int main()
{
        test t1, t2;

        t1.setcode();
        t2.setcode();

        test :: showcount();    // accessing static function

        test t3;
        t3.setcode();

        test :: showcount();

        t1.showcode();
        t2.showcode();
        t3.showcode();

        return 0;
}
```

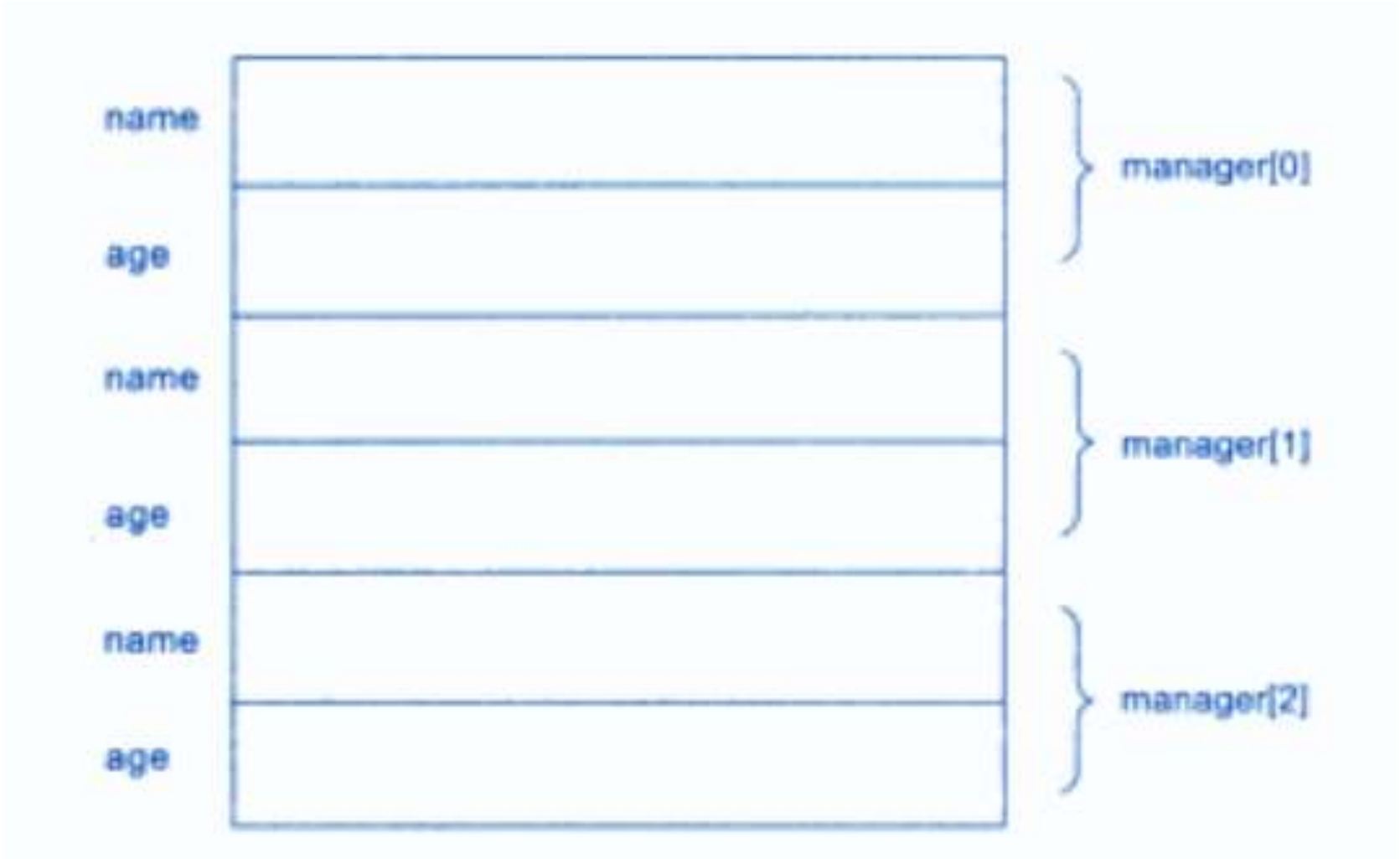
Output of Program 5.5:

```
count: 2
count: 3
object number: 1
object number: 2
object number: 3
```

Array of Objects

- Array of variables of type class can also be declared.
- These variables are called array of objects.

```
class employee
{
        char  name[30];       // string as class member
        float age;
  public:
        void getdata(void);
        void putdata(void);
};
void employee :: getdata(void)
{
        cout << "Enter name: ";
        cin  >> name;
        cout << "Enter age:  ";
        cin  >> age;
}
void employee :: putdata(void)
{
        cout << "Name: " << name << "\n";
        cout << "Age:  " << age  << "\n";
}
const int size=3;
```


name
age
name
age
name
age
manager[0]
manager[1]
manager[2]

```
int main()
{
        employee manager[size];
        for(int i=0; i<size; i++)
        {
                cout << "\nDetails of manager" << i+1 << "\n";
                manager[i].getdata();
        }
        cout <<  "\n";
        for(i=0; i<size; i++)
        {
                cout << "\nManager" << i+1 << "\n";
                manager[i].putdata();
        }
        return 0;
}
```

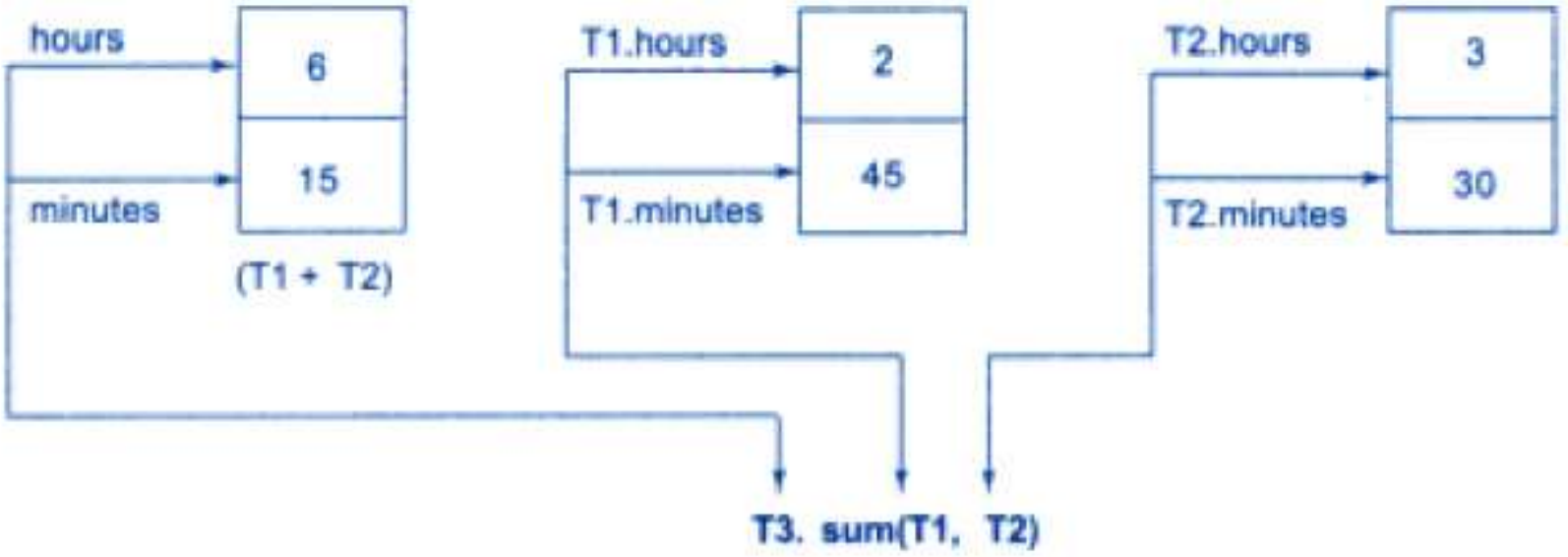
Objects as function arguments

```
#include <iostream>

using namespace std;

class time
{
        int  hours;
        int  minutes;
  public:
        void gettime(int h, int m)
        {  hours = h; minutes = m; }
        void puttime(void)
        {
                cout << hours << " hours and ";
                cout << minutes << " minutes " << "\n";
        }
        void sum(time, time);  // declaration with objects as arguments
};
void time :: sum(time t1, time t2)          // t1, t2 are objects
{
        minutes = t1.minutes + t2.minutes;
        hours = minutes/60;
        minutes = minutes%60;
        hours = hours + t1.hours + t2.hours;
}
```

```
int main()
{
        time T1, T2, T3;

        T1.gettime(2,45);     // get T1
        T2.gettime(3,30);     // get T2

        T3.sum(T1,T2);// T3=T1+T2

        cout << "T1 = "; T1.puttime();       // display T1
        cout << "T2 = "; T2.puttime();       // display T2
        cout << "T3 = "; T3.puttime();       // display T3

        return 0;
}
```

```
T1 = 2 hours and 45 minutes
T2 = 3 hours and 30 minutes
T3 = 6 hours and 15 minutes
```

Accessing members of objects within a called function



End of Unit-I