INDEX

UNIT-1 OOPS Introduction

1.1. Overview of Object Oriented Programming:

1.1.1. Introduction to Object Oriented Programming:

Object-oriented programming, as the term proposes uses objects in programming. The main purpose of Object-oriented programming (OOP) is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function. Object-oriented programming purposes to implement real-world entities like inheritance, hiding, polymorphism, etc in programming.

An object-oriented programming language (OOPL) is a high-level programming language based on the object-oriented model. Many modern programming languages are object-oriented, however some older programming languages, such as Pascal, do offer object-oriented versions. Examples of object-oriented programming languages include Java, C++ and Smalltalk.

1.1.2. Procedure Oriented and Object Oriented:

➢ **Procedure Oriented Programming (POP)**

- In POP, program is divided into small parts called functions.
- POP does not have any access specified.
- POP does not have any proper way for hiding data so it is less secure.
- In POP, Overloading is not possible.
- In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.
- Examples of POP are C, VB, FORTRAN, and Pascal.

➢ **Object Oriented Programming (OOP)**

- In OOP, program is divided into parts called objects.
- OOP has access specifiers named Public, Private, Protected, etc.
- OOP provides Data Hiding so provides more security.
- In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
- In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
- Example of OOP are C++, JAVA, VB.NET, C#.NET

1.1.3. Difference between C and C++:

C++ is derived from the C language. Other way C++ is a superset of C: Almost every correct statement in C is also a correct statement in C++, although the reverse is not true. Essential features added to C to create C++ concern classes, objects, and object-oriented

programming. (C++ was initially called "C with classes."). C++ has many other new features as well, including an enhanced method to input/output (I/O) and a new way to write comments. Figure 1.1 shows the association of C and C++.

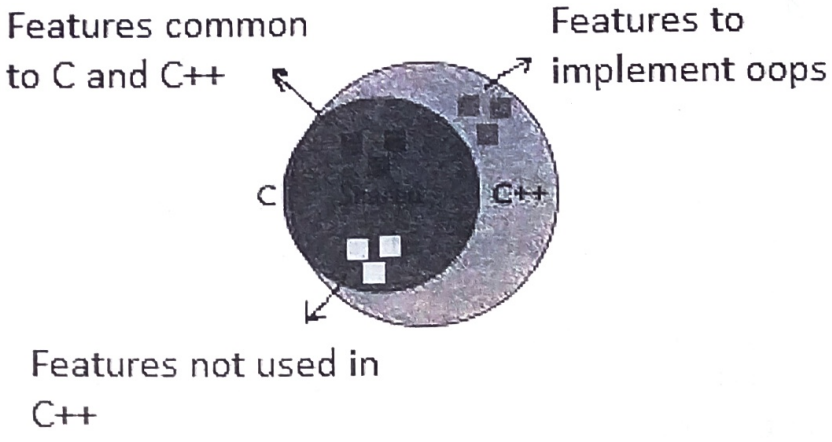

Fig 1.1 shows the association of C and C++

1.1.4. C++ Output / Input:

To execute input and output operations C++ uses an expedient abstraction called streams, in sequential media such as the screen, the keyboard or a file. A stream is an entity where a program can either insert or extract characters to/from. Header files available in C++ for Input – Output operation are:

iostream: iostream stands for standard input output stream. This header file contains definitions to objects like cin, cout, cerr etc shown in table 1.1.

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

Table: 1.1 shows the association of C and C++

iomanip: iomanip stands for input output manipulators. The methods declared in this files are used for manipulating streams. This file contains definitions of setw, setprecision etc.

fstream: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

- **Standard output (cout)**

In C++ program, the standard output by default is the screen, and the C++ stream object defined to access it is cout.

Example:

```
cout << "Output sentence"; // prints Output sentence on
screen
cout << 420;               // prints number 420 on screen
cout << x;                 // prints the value of x on
screen
cout << "Hello";  // prints Hello
cout << Hello;    // prints the content of variable Hello
cout << "First sentence.\n"; //prints First sentence.
with new line
cout << "Second sentence.\nThird sentence."; //prints
Second sentence and in new line Third sentence.
```

- **Standard input (cin)**

In C++ program, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin.

Example :

```
int age;
cin >> age;
```

1.1.5. Keywords in C++:

C++ programs have keyword as identifier, its reserved words in C++ library and used to perform an internal operation shown in table 1.2.

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	TRUE
break	export	protected	try
case	extern	public	typedef
catch	FALSE	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void

delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

Table: 1.2 Keywords C++

1.1.6. New style of header file specification:

#include <iostream>

It starts with a number sign (#). It's called a pre-processor directive. Header files contain definitions of Functions and Variables, which is imported or used into any C++ program by using the pre-processor #include statement. Header file have an extension ".h" which contains C++ function declaration and macro definition.

Each header file contains information (or declarations) for a particular group of functions. Like stdio.h header file contains declarations of standard input and output functions available in C++ which is used for get the input and print the output. Similarly, the header file math.h contains declarations of mathematical functions available in C++.

Types of Header Files in C++:

- System header files: It is comes with compiler.
- User header files: It is written by programmer.

```
#include<iostream>
int main()
{
    using namespace student;
    cout << "Hello, world!" << endl;
    return 0;
}
```

In above program print message on scree hello world! by using cout but we don't define cout here actually already cout has been declared in a header file called iostream.

1.1.7. Comments in C++:

The code ignored by the compiler which allows the user to make simple notes in the relevant areas of the source code using comments. Comments come either in block form or as single lines.

➢ **Single-line comments:** Start with // and continue until the end of the line. If the last character in a comment line is a \ the comment will continue in the next line.

Example:

```
// This is a single one line comment or
```

```
if (expression) // This needs a comment
{
statements;
}
else
{
statements;
}
```

The backslash is a continuation character and will continue the comment to the following line:

// This comment will also comment the following line \
std::cout << "This line will not print" << std::endl;

- **Multi-line comments:** (informally, *C style*), start with /* and end with */

Combining multi-line comments (/* */) with c++ comments (//) to comment out multiple lines of code:

Example:

```
/*
void Eventsum();
void Eventsum();
void Eventsum();
void Eventsum();
void Eventsum();
//*/
```

1.1.8. Variables in C++:

A variable is a name which is associated with a value that can be changed. For example, int n=50; here variable name is n which is associated with value 50, int is a data type that represents that this variable can hold integer values.

Syntax of declaring a variable:

data_type variable1_name = value1, variable2_name = value2;

For example:

int n1=20, n2=50;

Variables can be categorised based on their data type. For example, in the above example we have seen integer type's variables. Following are the types of variables available in C++.

int: These type of variables holds integer value.

char: holds character value like 'c', 'F', 'B', 'p', 'q' etc.

bool: holds boolean value true or false.

double: double-precision floating point value.

float: Single-precision floating point value.

Syntax of declaring a variable:

type variable_list;

For example:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

1.1.9. Reference Variables in C++:

A reference variable is an alternative name (alias), that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable. A reference variable must be initialized at the time of declaration.

Declaration

[data_type] & [reference_variable]=[regular_variable];

- Where, regular_variable is a variable that has already initialized, and reference_variable is an alternative name (alias) to represent the variable regular_variable.

Example:

```
int student_age=10;
int &age=student_age;
#include <iostream.h>
int main()
{
   int student_age=18;
   int &age=student_age;     // reference variable
   cout<< " value of student_age :"<< student_age << endl;
   cout<< " value of age :"<< age << endl;
   age=age+10;
   cout<<"\nAFTER ADDING 15 INTO REFERENCE VARIABLE \n";
   cout<< " value of student_age :"<< student_age << endl;
```

```
    cout<< " value of age :"<< age << endl;
    return 0;
}
```

1.1.10. The bool Data type:

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

Example

```
bool isCoding = true;
bool isFish = false;
cout << isCoding;  // Outputs 1 (true)
cout << isFish;  // Outputs 0 (false)
```

1.1.11. Importance of function prototyping in C++:

Prototype of a function is also called signature of the function. Function prototyping is a function declaration statement that tells the compiler about the return type of the function and the number as well as type of arguments required by the function at the time of calling it.

Syntax:

return_type function_name(type1 arg1, type 2 arg2, ...);

Advantages of function prototype:

- It helps the compiler in determining whether a function is called correctly or not. Each time when a function is called, its calling statement is compared with its prototype. In case of any mismatch, compiler reports an error.
- A function must be defined before calling it. But prototyping allows a function to be called before defining it.

1.1.12. Function Overloading:

Function overloading is a C++ programming feature that allows us to have more than one function having same name but different parameter list. For example the parameters list of a function my_func(int a, float b) is (int, float) which is different from the function my_func(float a, int b) parameter list (float, int). Function overloading is a compile-time polymorphism.

Function overloading is a feature in C++ where two or more functions can have the same name but different parameters.

C++ example to demonstrate function overloading.

```
#include <iostream>
using namespace std;
void print(int i) {
cout << " Here is int " << i << endl;
}
void print(double f) {
cout << " Here is float " << f << endl;
}
void print(char const *c) {
cout << " Here is char* " << c << endl;
}
int main() {
print(10);
print(10.10);
print("ten");
return 0;
}
```

Output:

Here is int 10

Here is float 10.1

Here is char* ten

1.1.13. Default Arguments:

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument with a default value.

Following is a simple C++ example to demonstrate the use of default arguments. We don't have to write 3 sum functions, only one function works by using default values for 3rd and 4th arguments.

C++ example to demonstrate default arguments.

```
#include<iostream>
using namespace std;
// A function with default arguments, it can be called with
2 arguments or 3 //arguments or 4 arguments.
int sum(int x, int y, int z=0, int w=0)
{
```

```
    return (x + y + z + w);
}
/* Driver program to test above function*/
int main()
{
    cout << sum(10, 15) << endl;
    cout << sum(10, 15, 25) << endl;
    cout << sum(10, 15, 25, 30) << endl;
    return 0;
}
```

Output:

25

50

80

Key Points:

- Default arguments are different from constant arguments as constant arguments can't be changed whereas default arguments can be overwritten if required.
- Default arguments are overwritten when calling function provides values for them. During calling of function, arguments from calling function to called function are copied from left to right.
- Once default value is used for an argument in function definition, all subsequent arguments to it must have default value. It can also be stated as default arguments are assigned from right to left.

1.1.14. Inline Function:

C++ inline function is powerful concept that is commonly used with classes. A function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time. To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line. A function definition in a class definition is an inline function definition, even without the use of the inline specifier. C++ example to demonstrate Inline Function.

```
using namespace std;
inline int Max(int x, int y) {
   return (x > y)? x : y;
}
```

```
// Main function for the program
int main() {
   cout << "Max (20,10): " << Max(20,10) << endl;
   cout << "Max (0,200): " << Max(0,200) << endl;
   cout << "Max (100,1010): " << Max(100,1010) << endl;
   return 0;
}
```

Output:

Max (20,10): 20

Max (0,200): 200

Max (100,1010): 1010

1.1.15. Scope Resolution Operator:

The :: (scope resolution) operator is used to get hidden names due to variable scopes so that you can still use them. The scope resolution operator can be used as both unary and binary. Two different kinds of scope are important here: local and file.

- Variables with local scope are visible only within a block.
- Variables with file scope are visible throughout a file.

A block is basically the code between an opening brace and a closing brace. Thus a function body is a block.You can use the unary scope operator if a namespace scope or global scope name is hidden by an explicit declaration of the same name in a block or class. For example, if you have a global variable of name my_var and a local variable of name my_var, to access global my_var, you'll need to use the scope resolution operator. For example,

```
#include <iostream>
using namespace std;
int my_var = 0;
int main(void) {
   int my_var = 0;
   ::my_var = 10;  // set global my_var to 1
   my_var = 20;    // set local my_var to 2
   cout << ::my_var << ", " << my_var;
   return 0;
}
```

Output:

10, 20

1.2. Classes And Object:

1.2.1. Structures in C:

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

Limitations of C Structures

- The C structure does not allow the struct data type to be treated like built-in data types:
- We cannot use operators like +,- etc. on Structure variables.

1.2.2. Structure in C++:

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types. For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store this information separately.

The struct keyword defines a structure type followed by an identifier (name of the structure).Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct person
{
    char name[30];
    int age;
    float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

```
#include <iostream>
using namespace std;
struct Person
{
    char name[50];
    int age;
    float salary;
};
int main()
{
    Person p1;
```

```
    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;
    return 0;
}
```

Output:

Enter Full name: Magdalena Dankova

Enter age: 27

Enter salary: 1024.4

Displaying Information.

Name: Magdalena Dankova

Age: 27

Salary: 1024.4

1.2.3. Access Specifier:

Access modifiers are used to implement an important feature of Object Oriented Programming known as Data Hiding. Access modifiers or Access Specifiers in a class are used to set the accessibility of the class members. That is, it sets some restrictions on the class members not to get directly accessed by the outside functions. There are 3 types of access modifiers available in C++: Public, Private, Protected

- Public: All the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
- Private: The class members declared as private can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.

- Protected: Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass (derived class) of that class.

Example of access modifiers:

```
class Person
{
public://access control
        string firstName;//these data members
        string lastName;//can be accessed
        tm dateOfBirth;//from anywhere
protected:
        string phoneNumber;//these members can be accessed inside this class,
        int salary;// by friend functions/classes and derived classes
private:
        string addres;//these members can be accessed inside the class
        long int insuranceNumber;//and by friend classes/functions
};
```

1.2.4. Classes:

A class is an abstract data type similar to 'C structure'. The Class representation of objects and the sets of operations can be applied to such objects. The class consists of Data members and methods. The primary purpose of a class is to hold data/information. This is achieved with attributes which are also known as data members.

The member functions determine the behaviour of the class i.e. provide a definition for supporting various operations on data held in form of an object.

Class class_name //Class is keyword and user-defined name

{ Access specifier: //can be privare, public or protected

  Data Members; //Variables to be used

  Methods; //Methods to access data members

};

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

1.2.5. Objects in C++:

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types.

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

Syntax:

ClassName ObjectName;

Accessing data members and member functions: The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is obj and you want to access the member function with the name CarName() then you will have to write obj.CarName()

// C++ program to demonstrate of Objects and accessing of data members

```
#include <iostream>
using namespace std;
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
int main() {
   Box Box1;        // Declare Box1 of type Box Class
   Box Box2;        // Declare Box2 of type Box Class
   double volume = 0.0;      // Store the volume of a box
here
   // box 1 specification
   Box1.height = 6.0;
   Box1.length = 7.0;
   Box1.breadth = 8.0;
   // box 2 specification
   Box2.height = 11.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;
   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;
   // volume of box 2
```

```
    volume = Box2.height * Box2.length * Box2.breadth;
    cout << "Volume of Box2 : " << volume <<endl;
    return 0;
}
```

Output:

Volume of Box1 : 336

Volume of Box2 : 1716

1.2.6. Characteristics of Access Specifier:

A key feature of object oriented programming is data hiding. It means that data is hidden. So that it can't be accessed mistakenly by functions outside the class. The class feature of C++ implements the concept of encapsulation by providing access specifier.

Access specifier is the keyword used for defining the scope of the member of class. There are three access specifier in C++. Private, Public, Protected

These three access specifier are used to specify the three levels of access protections for hiding data and function members.

Private:

The private data and function can be accessed from within the class. i.e. only from member of the same class. They are not accessible to the outside world. Hence, the primary mechanism for hiding data is to put data and function in a class and make them private.

Public:

Public members of a class are accessible from within or outside the class i.e. they can be accessed by any unction inside or outside the class. These public members are accessible from derived class and also from object outside the class.

Protected:

The protected data and functions can be accessed by the member functions of the same class. Also, these functions can be accessed by the member function of the derived class. But, it is not accessible to the outside world.

1.2.7. Function outside a class:

The functions are defined outside the class however they are declared inside the class. Functions should be declared inside the class to bound it to the class and indicate it as it's member but they can be defined outside of the class. To define a function outside of a class, scope resolution operator :: is used.

Syntax for declaring function outside of class

class class_name

{

```
  .....
  .....
  public:
   return_type function_name (args); //function declaration
};
//function definition outside class
return_type class_name :: function_name (args)
{
  .....; // function definition
}
```

Example:

```
#include <iostream>
using namespace std;
class car
{
  private:
    int car_number;
    char car_model[10];
  public:
    void getdata(); //function declaration
    void showdata();
};
// function definition
void car::getdata()
 {
   cout<<"Enter car number: "; cin>>car_number;
   cout<<"\n Enter car model: "; cin>>car_model;
 }
void car::showdata()
 {
   cout<<"Car number is "<<car_number;
   cout<<"\n Car model is "<<car_model;
 }
// main function starts
```

```
int main()
 {
    car c1;
    c1.getdata();
    c1.showdata();
    return 0;
 }
```

Output

Enter car number : 9999

Enter car model : Sedan

Car number is 9999

Car model is Sedan

1.2.8. Initialization of variable in C++:

Variables are arbitrary names given to a memory location in the system. These memory location are addresses in the memory. Variables are names given to these memory locations.

Initialization of a variable is of two types:

Static Initialization: Here, the variable is assigned a value in advance. This variable then acts as a constant.

Dynamic Initialization: Here, the variable is assigned a value at the run time. The value of this variable can be altered every time the program is being run.

1.2.9. Arrow Operator:

The . (dot) operator and the -> (arrow) operator are used to reference individual members of classes, structures, and unions. The dot operator is applied to the actual object. The arrow operator is used with a pointer to an object. For example, consider the following structure –

```
struct Employee {
  char first_name[16];
  int  age;
} emp;
```

The (.) dot operator

To assign the value "Naz" to the first_name member of object emp, you would write something as follows –

```
strcpy(emp.first_name, "Naz");
```

The (->) arrow operator

If p_emp is a pointer to an object of type Employee, then to assign the value "Naz" to the first_name member of object emp, you would write something as follows –

strcpy(p_emp->first_name, "Naz");

The -> is called the arrow operator. It is formed by using the minus sign followed by a greater than sign.

1.2.10. 'this' Pointer:

C++ has access to its own address through an important pointer called this pointer. The 'this' pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object. Friend functions do not have a 'this' pointer, because friends are not members of a class. Only member functions have a 'this' pointer.

```
#include <iostream>
using namespace std;
class Box {
   public:
      // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;
      }
      double Volume() {
         return length * breadth * height;
      }
      int compare(Box box) {
         return this->Volume() > box.Volume();
      }
   private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};
int main(void) {
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
```

```
    Box Box2(8.5, 6.0, 2.0);     // Declare box2

    if(Box1.compare(Box2)) {
       cout << "Box2 is smaller than Box1" <<endl;
    } else {
       cout << "Box2 is equal to or larger than Box1" <<endl;
    }
      return 0;
}
```

Output:-

Constructor called.

Constructor called.

Box2 is equal to or larger than Box1

☺ ☺ ☺ ☺ ☺

Exercises

1. Difference between POP vs. OOP.
2. Difference between C vs. C++.
3. Write a short note on 'CIN' and 'COUT'.
4. Write a short note on reference variables.
5. Explain the importance of function prototyping.
6. Write a short note on function overloading.
7. Explain briefly the concept of default arguments.
8. Explain inline functions.
9. Explain the scope resolution operator.
10. How to give comment in C++?
11. How and when to declare a variable in C++?
12. Explain briefly structure in C++.
13. Define: Access Specifiers.

14. Which are the access specifiers available in C++? Explain them, also explain their characteristics.
15. Explain class with its syntax and example.
16. Write a short note on objects in C++.
17. How to define a function outside the class?
18. When to use arrow operators?
19. Write a short note on 'this' pointer.
20. Do as Directed.
    a. POP does not have any access specifier. (T/F)
    b. iostream stands for _____.
    c. Start with _____ and continue until the end of the line.
    d. A reference variable is an alternative name _____.
    e. _____ data type is used to declare a variable whose value will be set as true (1) or false (0).
    f. The scope resolution operator can be used as both unary and binary. (T/F)
    g. Access modifiers are used to implement an important feature of Object Oriented Programming known as _____.
    h. The _____ operator and the _____ operator are used to reference individual members of classes, structures, and unions.
    i. The 'this' pointer is an implicit parameter to all member functions.(T/F)
    j. Friend functions do not have a 'this' pointer, because friends are not members of a class. (T/F)

Answers:

a. False
b. input output stream header file
c. //
d. Varible
e. Boolean
f. True
g. Data Hiding.
h. (dot) operator and the -> (arrow)
i. True
j. True

☺ ☺ ☺ ☺ ☺