21UCU403 - OBJECT ORIENTED PROGRAMMING

UNIT - 1

C++ Basics

ANSWER KEY

Created by:
Ajo. S. S
I B.Sc CS 'A'

Section - 1

1. Differentiate POP and OOPs.

Answer:

OOPs	POPs
Object oriented.	Structure oriented.
Program is divided into objects.	Program is divided into functions.
Bottom-up approach.	Top-down approach.
Inheritance property is used.	Inheritance is not allowed.
It uses an access specifier.	It doesn’t use an access specifier.
Encapsulation is used to hide the data.	No data hiding.
Concept of virtual function.	No virtual function.
Object functions are linked through message passing.	Parts of the program are linked through parameter passing.
Adding new data and functions is easy.	Expanding new data and functions is not easy.
The existing code can be reused.	No code reusability.
Can be used for solving big problems.	Not suitable for solving big problems.
Example : C++, Java.	Example : C, Pascal.

2. List out the advantages of OOPs.

Answer:

Improved software-development productivity:

Object-oriented programming is modular, as it provides separation of duties in object-based program development. It is also extensible, as objects can be extended to include new attributes and behaviours. Objects can also be reused within and across applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques.

Improved software maintainability:

Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.

Faster development:

Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.

Lower cost of development:

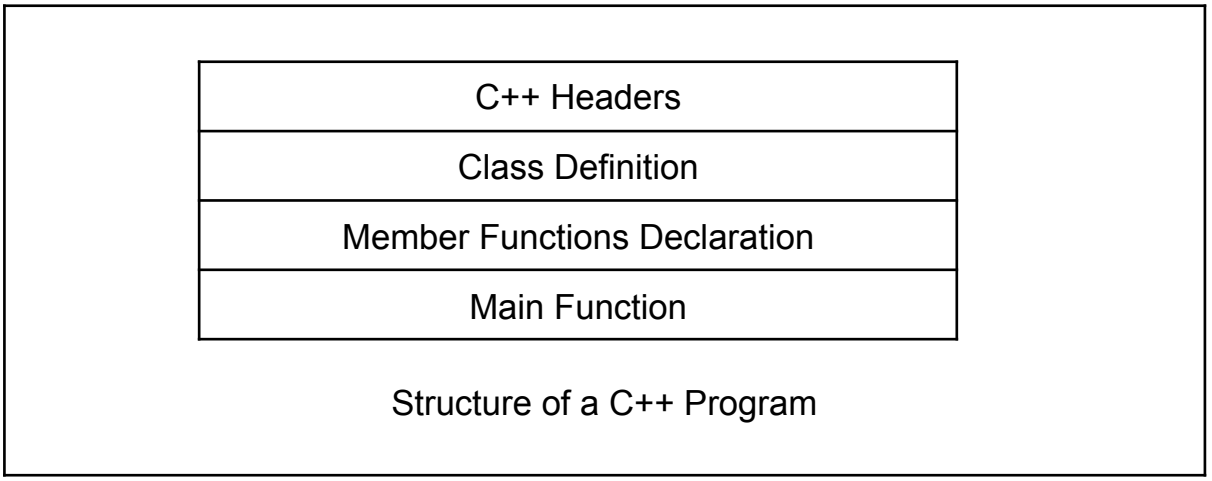
The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

Higher-quality software:

Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, object oriented programming tends to result in higher-quality software

3. Give note on structure of C++ Program.

Answer:



Structure of a C++ Program

Example:

```
#include <iostream>       // Header
using namespace std;
class age       // Class Definition
{
public:
   int number;
};
int main()       //Main Function
{
   age obj;
   cout << "Enter the Age :";
   cin >> obj.number;
   cout << obj.number << endl;
   return 0;
}
```

4. Explain about Simple functions in C++.

Answer:

A function is a block of code that performs a specific task.To perform any task, we can create a function. A function can be called many times. It provides modularity and code reusability.

There are two types of functions in C programming:

1. Library Functions: are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.

2. User-defined functions: are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

Declaring a function

Syntax:

```
returnType functionName (parameter1, parameter2,...)
{
   // function body
}
```

Example:

```
 // function declaration
void greet() {
   cout << "Hello World";
}
```

Here,

”void” is the return type.

“greet" is the function name.

5. Differentiate call by values and call by reference.

Answer:

Call by Value	Call by Reference
While calling a function, we pass values of variables to it. Such functions are known as “Call By Values”.	While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References.

In this method, the value of each variable in the calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function.
With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.	With this method, using addresses we would have access to the actual variables and hence we would be able to manipulate them.
In call-by-values, we cannot alter the values of actual variables through function calls.	In call by reference we can alter the values of variables through function calls.
Values of variables are passed by the Simple technique.	Pointer variables are necessary to define to store the address values of variables.

6. Give a note on call and return by reference.

Answer:

Call by reference in C++:

- In call by reference, the original value is modified because we pass reference (address).
- Here, the address of the value is passed in the function, so actual and formal arguments share the same address space.
- Hence, value changed inside the function, is reflected inside as well as outside the function.

Example:

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
int swap;
swap=*x;
*x=*y;
```

```
*y=swap;
}
int main()
{
int x=500, y=100;
swap(&x, &y);    // passing value to function
cout<<"Value of x is: "<<x<<endl;
cout<<"Value of y is: "<<y<<endl;
return 0;
}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

Section - 2

1. Explain C++ functions.

Answer:

A function is a block of code that performs a specific task.To perform any task, we can create a function. A function can be called many times. It provides modularity and code reusability.

There are two types of functions in C programming:
1. Library Functions: are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.

2. User-defined functions: are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

Declaring a function

Syntax:

```
returnType functionName (parameter1, parameter2,...)
{
```

```
    // function body
}
```

Example:

```
 // function declaration
void greet() {
    cout << "Hello World";
}
```

Here,
”void” is the return type.
“greet" is the function name.

Calling a function

In the above program, we have declared a function named greet(). To use the greet() function, we need to call it.

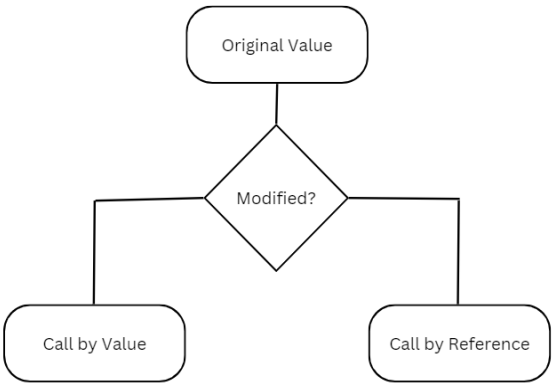
Here's how we can call the above greet() function.

```
int main()
{
     // calling a function
    greet();
}
```

There are two ways to pass value or data to function in C language:

- Call by value
- Call by reference

Original value is not modified in call by value but it is modified in call by reference.



2. Summarize on Macro vs Inline functions.

Answer:

Inline Function	Macro
These are functions provided by C++	Macros are preprocessor directives.
Inline keyword is used to declare the function as inline.	#define is used to declare the macro.
It can be defined inside or outside the class.	It cannot be declared inside the class.
Inline functions are parsed by the compiler.	Macros are expanded by the C++ preprocessor.
Inline functions can access the date member of the class.	Macros cannot access the data members of the class.
Compiler replaced the function call with the function code.	C preprocessor replaces every occurrence of macro template with its corresponding definition.
Inline function follows strict parameter type checking.	Macros do not follow parameter type checking.

Inline functions may or may not be expanded by the compiler. It depends upon the compiler's decision whether to expand the functions inline or not.	Macros are always expanded.
Can be used for debugging a program.	Cannot be used for debugging as they are expanded at pre-compile time.

inline int sum (int a, int b)
{
return (a+b);
}
	#define SUM (a,b) (a+B).

3. Define Translation.

Answer:

- A translation unit is any preprocessed source file. A translation unit is the basic unit of compilation in C++.
- This unit is made up of the contents of a single source file after it passes through preprocessing.
- It includes any header files without blocks that are ignored using conditional preprocessing statements like ifdef, ifndef, etc.
- A single translation unit can be compiled into an object file, library, or executable program.

In a C++ program, a symbol, for example a variable or function name, can be declared any number of times within its scope. However, it can only be defined once. This rule is the "One Definition Rule" (ODR). A declaration introduces (or reintroduces) a name into the program, along with enough information to later associate the name with a definition. A definition introduces a name and provides all the information needed to create it. If the name represents a variable, a definition explicitly creates storage and initialises it. A function definition consists of the signature plus the function body. A class definition consists of the class name followed by a block that lists all the class members. (The bodies of member functions may optionally be defined separately in another file.)

The following example shows some declarations:

```
int i;
int f(int x);
class C;
```

The following example shows some definitions:

```
int i{42};
int f(int x){ return x * i; }
class C {
public:
  void DoSomething();
};
```

4. Outline on default arguments.

Answer:

- 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 the calling function provides values for them. For example, calling the function sum(10, 15, 25, 30) overwrites the values of z and w to 25 and 30 respectively.
- When a function is called, the arguments are copied from the calling function to the called function in the order left to right. Therefore, sum(10, 15, 25) will assign 10, 15, and 25 to x, y, and z respectively, which means that only the default value of w is used.
- Once a default value is used for an argument in the function definition, all subsequent arguments to it must have a default value as well. It can also be stated that the default arguments are assigned from right to left. For example, the following function definition is invalid as the subsequent argument of the default variable z is not default.

Example program with default arguments:

```
#include <iostream>
using namespace std;
int sum(int x, int y, int z = 0, int w = 0) //assigning default values to z,w as 0
```

```
{
    return (x + y + z + w);
}
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

5. Write about friend functions in detail.

Answer:

- A friend function can be granted special access to private and protected members of a class in C++.
- They are the non-member functions that can access and manipulate the private and protected members of the class for they are declared as friends.
- A friend function can be:
    1. A global function
    2. A member function of another class
- Syntax:

```
friend return_type function_name (arguments);   //global function
                                or
friend return_type class_name::function_name (arguments);
//member function of another class
```

Features of Friend Functions:

- A friend function is a special function in C++ that in spite of not being a member function of a class has the privilege to access the private and protected data of a class.

- A friend function is a non-member function or ordinary function of a class, which is declared as a friend using the keyword “friend” inside the class. By declaring a function as a friend, all the access permissions are given to the function.
- The keyword “friend” is placed only in the function declaration of the friend function and not in the function definition or call.
- A friend function is called an ordinary function. It cannot be called using the object name and dot operator. However, it may accept the object as an argument whose value it wants to access.
- A friend function can be declared in any section of the class i.e. public or private or protected.

Advantages of Friend Functions:

- A friend function is able to access members without the need of inheriting the class.
- The friend function acts as a bridge between two classes by accessing their private data.
- It can be used to increase the versatility of overloaded operators.
- It can be declared either in the public or private or protected part of the class.

Disadvantages of Friend Functions:

- Friend functions have access to private members of a class from outside the class which violates the law of data hiding.
- Friend functions cannot do any run-time polymorphism in their members.

6. Discuss on virtual functions.

Answer:

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.
- The resolving of function calls is done at runtime.

Rules for Virtual Functions:

1. Virtual functions cannot be static.
2. A virtual function can be a friend function of another class.
3. Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
4. The prototype of virtual functions should be the same in the base as well as derived class.
5. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
6. A class may have a virtual destructor but it cannot have a virtual constructor.

Limitations of Virtual Functions:

Slower: The function call takes slightly longer due to the virtual mechanism and makes it more difficult for the compiler to optimize because it does not know exactly which function is going to be called at compile time.

Difficult to Debug: In a complex system, virtual functions can make it a little more difficult to figure out where a function is being called from.

Section - 3

1. Summarize on the components of C++ Functions.

Answer:
A function is a set of statements that take inputs, do some specific computation, and produce output. The idea is to put some commonly or

repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

Function Declaration

- A function declaration tells the compiler about the number of parameters function takes data-types of parameters, and returns the type of function.
- Putting parameter names in the function declaration is optional in the function declaration, but it is necessary to put them in the definition.
- Below are examples of function declarations. (parameter names are not there in the below declarations)

Syntax:

```
returnType functionName (parameter1, parameter2,...)
{
   // function body
}
```

Example:

```
 // function declaration
void greet() {
   cout << "Hello World";
}
```

Here,
”void” is the return type.
“greet" is the function name.

Function Definition

Program to show function definition:

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

void fun(int x)
{
```

```
    // definition of function
    x = 30;
}

int main()
{
    int x = 20;
    fun(x);
    cout << "x = " << x;
    return 0;
}
```

Calling a function

In the above program, we have declared a function named greet(). To use the greet() function, we need to call it.

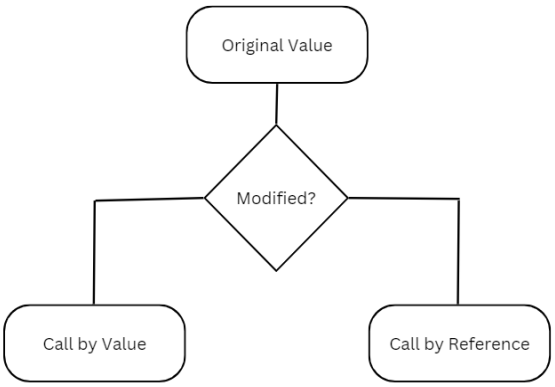
Here's how we can call the above greet() function.

```
int main()
{
     // calling a function
    greet();
}
```

There are two ways to pass value or data to function in C language:

- Call by value
- Call by reference

Original value is not modified in call by value but it is modified in call by reference.



2. Discuss on class and objects declaration inside and outside the class.

Answer:

- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.
- An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Defining Class and Declaring Objects

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

Syntax:

```
class class_name
{
   Access specifier;    //can be public, private and protected
   Data members;     //variables to be used
   Member function ()     //methods to access data members
   {
       //statements
    }
};    //ends with a semicolon
```

Declaring Objects:

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

Accessing data members and member functions:

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

Member Functions in Classes

There are 2 ways to define a member function:

1. Inside class definition
2. Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

Syntax:

```
class class_name
{
  //statements
  Return_type class_name :: function_name (parameter_list)
}
Return_type class_name :: function_name (parameter_list)
```

3. What is overloading of functions? Discuss in detail.

Answer:

- Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters.
- When a function name is overloaded with different jobs it is called Function Overloading.
- In Function Overloading “Function” names should be the same and the arguments should be different.
- Function overloading can be considered as an example of a polymorphism feature in C++.

The parameters should follow any one or more than one of the following conditions for Function overloading:

1. Parameters should have a different type:
   Example: add(int a, int b)
   add(double a, double b)

Example Program:

```
#include <iostream>
using namespace std;
void add(int a, int b)
{
  cout << "sum = " << (a + b);
}
void add(double a, double b)
{
    cout << endl << "sum = " << (a + b);
```

```
}
int main()
{
    add(10, 2);
    add(5.5, 6.5);
    return 0;
}
```

Output:

```
sum = 12
sum = 12
```

2. Parameters should have a different number

   Example: add(int a, int b)
   add(int a, int b, int c)

Example Program:

```
#include <iostream>
using namespace std;
void add(int a, int b)
{
  cout << "sum = " << (a + b);
}
void add(int a, int b, int c)
{
    cout << endl << "sum = " << (a + b + c);
}
int main()
{
    add(10, 2);
    add(5, 6, 4);
    return 0;
}
```

Output:

```
sum = 12
sum = 15
```

3. Parameters should have a different sequence of parameters.
   Example: add(int a, double b)
   add(double a, int b)

Example Program:

```
#include<iostream>
using namespace std;
void add(int a, double b)
{
	cout<<"sum = "<<(a+b);
}
void add(double a, int b)
{
	cout<<endl<<"sum = "<<(a+b);
}
int main()
{
	add(10,2.5);
	add(5.5,6);
	return 0;
}
```

Output:

```
sum = 12.5
sum = 11.5
```