18/02 Excellent (A+)

Assignment - 1

Q1. What is the need for object oriented programming ?

→ 1. To solve real time problem.
2. Emphasis is on data rather than procedure.
3. Programs can be divided into known objects.
4. Objects are used wrap data and functions together.
5. Object needs to communicate with each other.
6. Data hiding and security.
7. Data structures are designed in characterize object.
8. Follows bottom up approach.

Q2. Write a short note on :-

a Namespace :-

→ Namespace provide the space where we can define or declare identifies i.e variable method classes.
- Using namespace we can define the space or context in which identifiers are defined i.e variables, method classes in essense of namespace defines a scope.
- A namespace is designed to overcome the difficulty of similiar name functions, classes variables available in different libraries.
- Eg:- Standard library in C++ - std.
the keyword 'using' is used to introduce the namespace.
using namespace std;

b. Objects :-

→ - An object is an instance of a class.
- They are basic run time entities in object oriented progr..
- In c++ class variables are ~~class~~ called objects. Using objects we can access the member variables and

member function of class.
- When a class is defined, no memory is created but when it is initiated (i.e an object is created) memory is allocated. A single class can create many objects.

Syntax:- class_name Object_name ;
eg :- Fruit f1; // Fruit= class f1 = object.

c) Classes :-
→ - Class is a user defined data type and comprises data and function together.
- Class is a black box and has logical existence.
- It is a basic feature of OOP language
- In C++ classes, 3 kinds of access specifier are used:
1. Private 2. Public 3. Protected.
- Member variables and member methods of a class can be accessed using objects.
- Syntax:-

```
class class_name
{ private : //
  protected : //
  public : //
}
```

Eg:-

```
class fruit
{ private :
    a = 1
  protected :
    b = 2
  public :
    c = 3
}.
```

d) Data members :-
→ - Data members are that variables that are declared within the class.
- These members are declared within along with data type.

- The <u>access specifiers</u> to these members can be private, public or protected.
- These data members can be accessed by the <u>main function</u> using the <u>object</u> of the class.

<u>eg:-</u>
```
class rectangle {
{ private :
     int len, br
  public :
        void area ()
}
```
} — data members.

e) Methods:-

→ - Methods are <u>functions</u> that belongs to the class.
- There are two ways to define functions that belong to a class : a. <u>Inside</u> class defined b. <u>Outside</u> class defined.
- You can access methods by creating an <u>object</u> of class and using the dot <u>synatax (.)</u>.

<u>eg:-</u> Inside class definition.

```
class myClass        // class
{ public :           // access specifier
         int a ;  // data member
         void myMethod ()
         { cout << "Hello" ;

}
```

Outside class definition

```
class myClass
{ public :
   void methed_2 ()
}
void myClass:: method_2 ()
{ cout << "Hello" ;
}
```

f) Messages :-

→ - Objects communicate with each other by <u>sending</u> and <u>recieving information</u> to each other.
- A message for an object is a <u>request</u> for <u>execution</u> of a produre and therefore will invoke

a function in the recieving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

g) Data encapsulation:-
→ - In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit.
- In OOP, Encapsulation is defined as binding together the data and the functions that manipulate them.

h) Data Abstraction:-
→ - Data abstraction is another important feature of OOP.
- It refers to using very essential features without adding the background details are not given.
- It means only less number of details are added and implementation details are not given.
- It is a ways of information hiding from the outside world. Data abstraction also referred as abstract data types (ADT).
- Data abstracting and encapsulation are ways of information hiding.

i) Information hiding:-
→ Encapsulation supports information hiding by making use of the three access specifiers of a class.
- public: It can be used anywhere without the access restrictions

- private : It can be only used by the members and friends of class.
- protected: It can be only used by the members and friends of class and derived from them

Q3. List down 8 features of OOP and explain in detail.

→ 1. Encapsulation:

The binding up of data and related code into a single unit is called encapsulation. This keeps both the data and the code from being misused by the outerworld. It acts as the code wrapper which prevents the data from being accessed by outer code outside the wrapper.

Real time eg:- A car encapsulates the sterring, engine, etc.

Eg in C++:- A class encapsulate, data (state) and functions (behavior).

2. Polymorphism : (Poly - many , morph - forms).

The ability of an entity to take more than one form is called polymorphism. It allows two or more functions / operators to respond to the same message in different ways. The behaviour depends on the type of data used in the operation. It plays an imp. role in allowing different internal structures to share the same external interface. With polymorphism we can create a single method and apply it for different objects

eg:- A man is an employee at office and a father at home.

3. Data abstraction:

The act of representing essential features and

hiding all the complexities is known as data abstraction.
eg:- In a music system we just press the button and listen to the music without knowing any technical details of how the volume changes or the music is played.
When we call a function, only function is called and the task is performed, what does the function includes and how it is carried out is abstracted.

4. Inheritance:-
The capability of a class to derive properties and characteristics from another class is called as Inheritance. It is one of the most important features of OOP.

5. Modularity:-
Modularity is the concept of separating a program into different module. Each module is solved independently and all are then combined to form a single unit.

6 Persistency:-
Fixed memory space. eg:- Objects.

7. Concurrency:-
Concurrency occurs when multiple copies of a program run simultaneously while communicating with each other.

8. RTTI (Run Time Type Identification) :-
It is a mechanism which allows the type of object to be determined during program execution.

Q4. Explain benefits of OOP.

→ 1. Using inheritance the reductant code can be eliminated and existing classes can be used.
2. The standard working modules can be created using OOP. These modules can then communicate to each other to acomplish certain code.
3. Due to data hiding properties, important data can be kept away from unauthorized access.
4. It is possible to create multiple objects for a given class.
5. For upgrading the system for, small scale to large scale is possible due to OOP feature
6. Due to data centred nature of OOP, most of the details of application model can be captured
7. Message passing technique allows the object to communica--te to the external systems.
8. Positioning the code for simplicity, understanding, and debugging is possible due to object oriented and modular approach.

Q5. Which are the primitive data types used in C++ ?

→ Data types in C++ are divided into three types, primitive data types, derived data types and user defined data types.

Primitive data types are built-in or predefined data types which can be used directly by the user to declare variables.
Some of these are:

Data type	Keyword	Size (in bytes)
Integer	int	4
character	char	1
Boolean	bool	1
Floating-point	float	4
Double floating point	double	8
void, wide, etc		

eg:-
```
int    a ;
float  b = 10.2 ;
char   c = 'C' ;
```

```
# include <iostream>
using namespace std ;
int main ()
{
    int a, b ;
    float c= 6.987 ;
    char q = 'F' ;
    cout << "Enter values " << endl ;
    cin >> a >> b ;
    cout << " Sum " << a + b + c ;
    cout << q ;
    cout << " Memory req. : 13 bytes " ;
}.
```

Q6. What is meant by stoucture ? Write the syntax and example of it.

Ans. 1. A structure is a collection of variables referenced under one name, providing a convinient means of keeping related information together.

2. A structure is a decoration template that may be used to create structured object.

3. The variables that make up the structures are called the 'members'.

4. Generally, all of the members of a structure are logically related.

5. The 'struct' keyword is used to declare a structure in C++.

6. Syntax :-

```
struct structure_name
{
    // member declaration.
}.
```

7. eg:-

```
struct student
{
    class name [20] ;
    int id ;
    int age ;
}.
```

Q7. What is meant by enumeration ? Write the syntax and example of it.

Ans. 1. An enumeration is an set of named integer constants that specify all the legal values a variables of that legal values a variables of that type may have. They are common in everyday life.

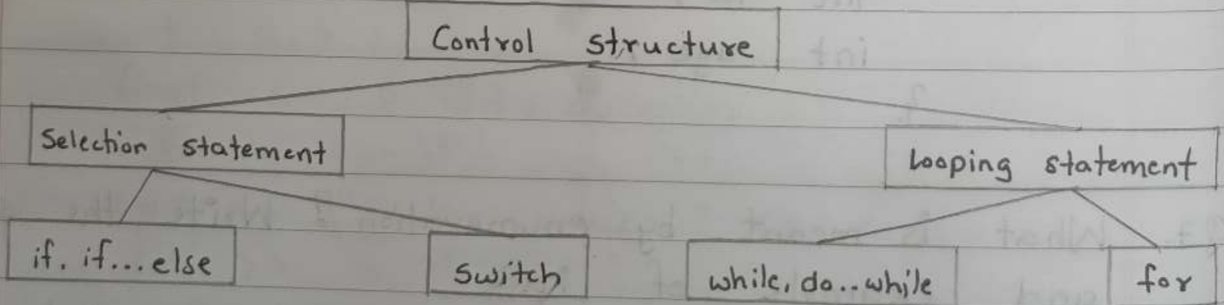
Enumerations are defined much like structures, but the keyword `enum' signals the start of an enumeration type.

2. Syntax :-
enum enum_type-name {enumeration list } variable list ;

3. Examples:-
enum coins { penny, nickel, dime, dollar } ;

4. Applications:- (i) It offers an easy way to work with sets of related constants.
(ii) To create set of constants of use with variables and properties.

Q8. Explain control structures using C++ in detail.
Ans. 1. It is used to control the flow of program.
2. Different control statement structures is used to control the flow the program.
3. It can be divide into selection statement and looping statement.
4. Selection statement include if, if...else, switch, etc.
5. Looping statement include while, do...while, for, etc.



Q9. Explain array and structure in detail.
Ans. 1. An array is a collection of variables of same type that are referred through common name. A specific element in an array can be accessed

through its index. It is a collection of homogeneous elements.

eg:-

```
int roll-no.[5] = {10,20,30,40,50}.
int main () {
            cout << roll-no.[3].
        }.
```

2. In this program, an array named 'roll-no' is declared it's size is 5, we can access the values stored in the array through index.
3. The above program will return 40 as output [index: 3].

b) Structure :-
1. Structure is collection of variable referred under one name, providing convinient means of keeping related information together.
2. The variable that make up structure are called 'members'.
3. Structure consists of heterogeneous elements.

eg:-

```
struct address {
            char name [2];
            char city [1];
            char state [5];
        }
  address . ad_obj ;
```

Q10 Which access specifiers are used in C++. Explain in detail.

Ans. 1. Access specifiers defines how the members of a class can be accessed.
2. In C++ there are three access specifiers :-

a. Public : Members are accessible from outside the class.
b. Private : Members cannot be accessed outside the class.
c. Protected : Not outside the class but in inherited classes.

3. Syntax :-

```
class   classname
{
 Private   ;
          //
 Public    ;
          //
 Protected ;
          //
};
classname   c_obj ;.
```

eg:-
```
class   student
{
  Private ;
           //Roll-no.
  Protected ;
           // Student - name
  Public  ;
           // College - name
};
Student   st-obj ;
```

4. Applications :- (i) To assign the accessibility to class members.
(ii) Used to hide / show data from others.

Q11. What is concept of interface and implementation.

Ans. In C++, the concept of interface and implementation related to the way classes are structured and designed.

a) Interface :

1. An interface is like a contract that defines a set of method declarations without providing their implementations.
2. In C++, interfaces are typically represented during abstract base classes.
3. Interfaces define what methods a class should have but they don't specify how these methods should be implemented.
4. Users of the interface only need to know what functions are available and their signatures, not the details of how they work.

b) Implementation:

1. It refers to the actual code that provide the functionality of a class or interface.
2. In C++, classes that implement an interface provided the concept implementation for all the methods declared in that interface.
3. Implementation may include data members, private methods and other details that are not part of the interface.

Q12. What is meant by function? Explain function prototype, function body and function calling in detail. How to access function and utility function.

Ans. A function is a reusable block of code that performs a specific task or a set of task. Functions are essential for modularizing code, making it more organized and easier to maintain.

1. Function Body: It is the actual implementation it contains a series of statements that perform the

intended task

Syntax:
```
type name (arg) {
        // body
}
```

Eg:-
```
int add (inta, int b) {
        return a + b
}
```

3. Function calling: To use a function we need to call it, function calling involves providing the required arguements and invoking the function by its name.
Syntax: func-name (arg);
Eg:- add (3,2);

4. Function prototype: It is not but the signature of function. It consists of name of functions, passing parameters to the functions, returns data type of functions.

5. Utility function: They are a type of function that provides commonly used functionality to simplify coding. They are often used to perform tasks like ~~inn~~ input/output, string manifulation, etc
Eg:- 'printf' function in C is a utility function for formatted o/p.

6. Applications:-
(i) Code Organization
(ii) Reusability
(iii) Abstraction.
(iv) Parameterization
(v) User Interface.
(vi) Recursion.
(vii) Modularization.

Q13. What is meant by constructor and destructors ? Which are the types of constructors ? Explain in detail.

Ans. 1. A constructor is a special member function with the same name as the class. It is used to initialize the object's data members and is automatically called when an object is ~~or~~ created.

Syntax:-

```
class_name ()
{
  // constructor's Body
}
```

2. Destructor: It is also a special member function with the same name as the class but prefixed with a ~~tiel~~ tilde (~). It is used to clean up resources and perform any necessary cleanup operations before an object is destroyed. The destructor is automatically called when an object goes out of scope or is explicity deleted.

Syntax:-

```
~ class_name ()
{
  // Destructor's Body
}
```

3. Types of constructors:-

a) Default constructor: It is a constructor that is automatically called when an object is. created with no arguements. It initialization is the objects data members to their default values.

Syntax:

```
class-name ()
{
  // construction Body
}
```

b). Parameterized constructor: It is a constructor that takes one or more parameters. It allows you to initialize the objects with specific values provided as arguements when creating object.

Syntax:-

```
class-name (arg1, arg2, ---)
{
 //constructor's Body
}
```

c) Copy constructor: It is a constructor that creates a new object by copying values from an existing object of same class. It is used to create a copy of an object and is called when an object is initialized with another object of same class.

Syntax:-

```
class_name (const class-name & other)
{
     // constructor's Body
}.
```

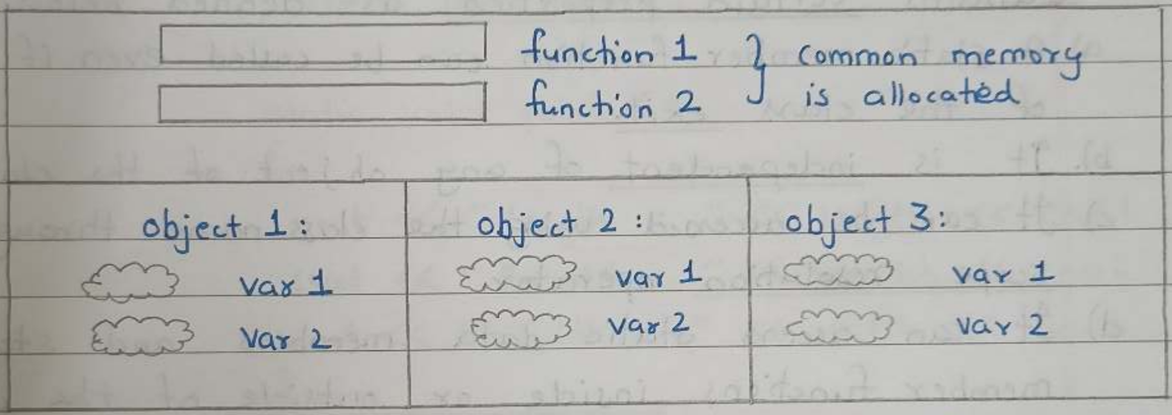
Q14. Explain object and memory requirements in detail.

Ans. 1. In memory space objects are allocation, when they are declared.

2. The member function are created and place in

the memory only when they defined as a part of class specification are defined as a part of class specification.

3. As the object are using the same member functions of belonging class, no seperate memory space is created for these function when object are created.
4. Only for member variables the seperate memory space is created for each object.



Memory for var 1 and var 2 is allocated for each object.

Dynamic memory allocation :-

1. It is the process that allocates memory for variables and data structures at runtime.
2. When the program requests it, this allows for flexibility and efficiency as the size and location of memory blocks can be changed according to the program logic and datasize.
3. We use new operator in C++ which defines two unary operator new and delete that perform the task of allocating and deleting memory during runtime, it also called as.

Syntax:-

New - pointer var = new data type ;

Delete - delete pointer - variable ;

Eg:- New - p = new int ;
Delete - delete p ;

Q15 Write short note on :-

a) Static member variable and function.

Ans. 1. Static member function in a class is the function that is declared as static because of which functions attains certain preperties as defined below;

a) A static member function can be called even if no object of the class exit.

b). It is independent of any object of the class.

c) It can be accessed using the class name through the scope resolution operator.

d) It can access static data members and static member functions inside or outside of the class.

e). They have access inside the class and cannot access the current object pointers.

f) It is also used to determine how many objects of the class have been created.

2. Static members are frequently used to store information that is shared by all objects in class.

3. For instance you may keep track of the quantity of newly generated objects of a specific class type using a static data member as a counter. This static data member can be interested each time an object is generated to keep track of the overall no. of objects.

Syntax:-

```
static returntype function_name ().
```

Eg:-

```
    # include <iostream>
    using namespace std;
    class Box
    {
      private :
              static int length ;
              Static int breadth ;
              static int width ;
      public :
             static void point ()
             {
              cout << " Length : " << length << endl ;
              cout << " Breadth : " << breadth << endl ;
              cout << " width : " << width << endl ;
             }
    };

    int Box :: length = 10 ;
    int Box :: breadth = 20 ;
    int Box :: heigth = 30 ;

int main ()
{
   Box b ;
   cout << " called through object name : \n " << endl ;
   b. print () ;

   cout << " \n Called through class name : \n << endl;
   Box :: print () ;
   return 0 ;
}
```

b) Inline function :-

Ans. 1. An inline function is one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a seperate set of instructions in memory.

Syntax:-

```
inline returntype function_name ().
```

Eg:-

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

inline int max (int a, int b)
{
  return a > b ? a : b;
}
int main ()
{
 cout << max (10,20);
 cout << " " << max (99,88);
 return 0;
}
```

2. Inline functions that are small have higher efficiency and better results than the lengthier inline functions.
3. High usage of inline function should be avoided to avoid bloating.

c) Friend function :

Ans. 1. These are functions which are not part of the class but it can access the private and protected members of the class.

2. They have special privilage to access the members of the class but not part of the class.
3. The prototype is same as that of normal function with friend keyword at the start of the function.

Syntax :-

```
friend returntype function_name () ;
```

4. They should be declared inside the class.
5. Definition of friend function is outside the class scope.

Eg:-

```
# include <iostream>
using namespace std ;
class friend_class
{
  private :
  int x
  public
  friend_class ()
      {
        x = 0
      }
      friend int myfunction (friend_class)
}
 int myfunction (friend_class c)
{
  c.x = 500 ;
   reture c.x ;
}
int main ()
{  friend class f ;
 cout << "\n Value of var is : " << myfunction (f) ;
 return 0
}
```