UNIT 2(THEORY SOLUTION)

1. **What sorts of shortcomings of structure are addressed by classes? Explain giving appropriate example.[PU:2014 fall]**

   The shortcoming of structure that are addressed by classes are as follows:

   1) Class can create a subclass that will inherit parent's properties and methods, whereas Structure does not support the inheritance.
   2) Structure members can be easily accessed by the structure variables in their scope but class members doesn't due to feature of data hiding.
   3) Classes support polymorphism (late binding), whereas structure doesn't.
   4) Also we can use "this" pointers in classes due to which you don't have to explicitly pass an object to a member function.
   5) The structure data type cannot be treated like built in types while performing arithmetic operations. But it is valid in class using the concept of operator overloading.
      For example:

      ```
      struct complex
      {
      int real;
      int imag;
      }
      struct complex c1,c2,c3;
      c3=c1+c2; Is illegal .
      ```

2. Differentiate between structure and class. Why Class is preferred over structure? Support your answer with suitable examples.*[PU:2016 fall]*

Structure	Class
1. A structure is a collection of variables of different data types under a single unit.	1. A class is a user-defined blueprint or prototype from which objects are created.
2. To define structure we will use "struct" keyword.	2. To define class we will use "class" keyword.
3. A structure has all members public by default.	3. A class has all members private by default.

4. A Structure does not support inheritance.	4. A Class can inherit from another class. Which means class supports inheritance.
5. Structures are good for small and isolated model objects.	5. Classes are suitable for larger or complex objects.
6. Structure is a value type and its object is created on the stack memory.	6. Class is a reference type and its object is created on the heap memory.
7. Function member of the struct cannot be virtual or abstract	7. Function member of the class can be virtual or abstract.

Due to the following reasons class is preferred over structure.

- Structures in C++ doesn't provide data hiding where as a class provides data hiding
- A Structure is not secure and cannot hide its implementation details from the end user while a class is secure and can hide its programming and designing details.
- Classes support polymorphism (late binding), whereas structure doesn't.
- Class support inheritance but structure doesn't.
- "this" pointer will works only with class.
- Class lets us to use constructors and destructors.

3. Where do you use friend function? What are the merits and di-merits of friend function?

In some situation non-member function need to access the private data of class or one class wants to access private data of second class and second wants to access private data of first class. This can achieve by using friend functions.
The non-member function that is "friendly" to a class, has full access rights to the private members of the class.

For example when we want to compare two private data members of two different classes in that case you need a common function which can make use of both the private variables of different class. In that case we create a normal function and make friend in both the classes, as to provide access of theirs private variables.

Merits of friend function:

- It can access the private data member of class from outside the class
- Allows sharing private class information by a non-member function.
- It acts as the bridge between two classes by operating on their private data's.

• It is able to access members without need of inheriting the class.
• It provides functions that need data which isn't normally used by the class.

Demerits of friend function:
• It violates the law of data hiding by allowing access to private members of the class from outside the class.
• Breach of data integrity
• Conceptually messy
• Runtime polymorphism in the member cannot be done.

4. Does friend function violate the data hiding? Explain Briefly.[PU:2017 fall]
The concept of data hiding indicates that nonmember functions should not be able to access the private data of class. But, it is possible with the help of a "friend function". That means, a function has to be declared as friend function to give it the authority to access to the private data.
Hence, friend function is not a member function of the class but can access private members of that class. So that's why friend function violate data hiding.

Let's illustrate the given concept with the following example.

```
#include<iostream>
using namespace std;
class sample
{
private:
int a,b;
public:
void setdata(int x,int y)
{
a=x;
b=y;
}
friend void sum(sample s);
};
void sum(sample s)
{
cout<<"sum="<<(s.a+s.b)<<endl;
}
int main()
{
sample s;
s.setdata(5,10);
sum(s);
return 0;
}
```

Here in class named sample there are two private data members a and b. The function sum() is not a member function but when it is declared as friend in class sample then sum() can access the private data a and b. which shows that friend function violate the concept of data hiding.

5. What are the limitations of static member functions

Limitations of static member functions:

- It doesn't have a "this" pointer.
- A static member function cannot directly refer to static members.
- Same function can't have both static and non static types.
- It can't be virtual.
- A static member function can't be declared as const or volatile.

6. Data hiding Vs Encapsulation

Data hiding	**Encapsulation**
1) Data Hiding means protecting the members of a class from an illegal or unauthorized access.	1) Encapsulation means wrapping the implementation of data member and methods inside a class.
2) Data hiding focus more on data security.	2) Encapsulation focuses more on hiding the complexity of the system.
3) The data under data hiding is always private and inaccessible.	3) The data under encapsulation may be private or public.
4) When data member of class are private only member function of that class can access it.	4) When implementation of all the data member and methods inside a class are encapsulated, the method name can only describe what action it can perform on an object of that class.
5) Data hiding is a process as well as technique.	5) Encapsulation is a sub-process in data hiding.

7. Abstraction Vs Data hiding

Abstraction	Data hiding
1) Abstraction is a mechanism of expressing the necessary properties by hiding the details.	1) Data Hiding means protecting the members of a class from an illegal or unauthorized access
2) It's purpose is to hide complexity.	2) It's purpose is to achieve encapsulation.
3) Class uses the abstraction to derive a new user-defined datatype	3) Data hiding is used in a class to make its data private.
4) It focuses on observable behavior of data.	4) It focuses on data security.

8. What is the difference between message passing and function call? Explain the basic message formalization. [PU:2006 spring]

Message Passing	Function Call
1) In a message passing there is a designated receiver for that message; the receiver is some object to which message is sent.	**1)** In function call, there is no designated receiver.
2) Interpretation of the message (that is method is used to respond the message) is determined by the receiver and can vary with different receivers.	**2)** Determination of which method to invoke is very early binding of a name to fragment in procedure calls
3) Message passing must involve name of the object, function name and information to be sent.	3) Simply function name and its arguments is used to made function call.
4) A message is always given to some object, called the receiver.	4) Name will be matched to a message to determine when the method should be executed Signature the combination of return type and argument types.
5) Example: st.getdata(2,5)	5) **Example:** getdata(2,5);

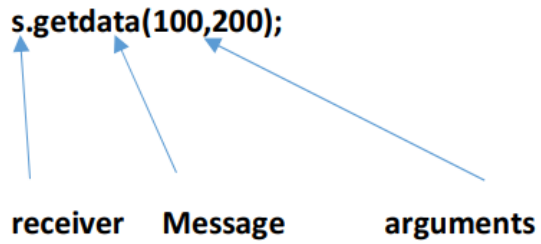
Message passing formalization

Message passing means the dynamic process of asking an object to perform a specific action.

- ✓ A message is always given to some object, called the receiver.
- ✓ The action performed in response to the message is not fixed but may be differ, depending on the class of the receiver. That is different objects may accept the same message and yet perform different actions.

There are three identifiable parts to any message-passing expression. These are

1) **Receiver:** the object to which the message is being sent
2) **Message selector:** the text that indicates the particular message is being sent.
3) **Arguments** used in responding to the message.



Example of message passing

```
#include<iostream>
using namespace std;
class student
{
int roll;
public:
void getdata(int x)
{
roll=x;
}
void display()
{
cout<<"Roll number="<<roll;
}
};
int main()
{
student s;
s.getdata(325); //objects passing message
s.display(); //objects passing message
return 0;
}
```

9. What are the possible memory errors in programming. [PU:2014 spring]

Memory errors occur very commonly in programming, and they can affect application stability and correctness. These errors are due to programming bugs. They can be hard to reproduce, hard to debug, and potentially expensive to correct as well. Applications that have memory errors can experience major problems.
Memory errors can be broadly classified into Heap Memory Errors and Stack Memory Errors.
Some of the challenging memory errors are:
1. **Invalid Memory Access in stack and heap**: This error occurs when a read or write instruction references unallocated or deallocated memory.
2**. Memory leaks**
Memory leaks occur when memory is allocated but not released.
3. **Mismatched Allocation/Deallocation**
This error occurs when a deallocation is attempted with a function that is not the logical counterpart of the allocation function used.
To avoid mismatched allocation/deallocation, ensure that the right de-allocator is called. In C++, new[] is used for memory allocation and delete[] for freeing up.
4. **Missing allocation**
This error occurs when freeing memory which has already been freed. This is also called "repeated free" or "double free".
5. Uninitialized Memory Access
This type of memory error will occur when an uninitialized variable is read in your application. To avoid this type of memory error, always initialize variables before using them.

10. Is it mandatory to use constructor in class. Explain?

Constructor is a 'special' member function whose task is to initialize the object of its class.
It is not mandatory to use constructor in a class. As we know, Constructor are invoked automatically when the objects are created.
So that object are initialized at the time of declaration. There is no need to separate function call to initialize the object. Which reduces coding redundancy and minimizes the programming error such as uninitialized memory access.

11. **Differentiate between constructor and destructor.**

Constructor	Destructor
1) Constructor is used to initialize the instance of the class.	1) Destructor destroys the objects when they are no longer needed.
2) Constructors is called when new instances of class is created.	2) Destructor is called when instances of class is deleted or released.
3) Constructor allocates the memory.	3) Destructor releases the memory.
4) Constructor can have arguments.	4) Destructor cannot have any arguments.
5) Overloading of constructor is possible.	5) Overloading of Destructor is not possible.
6) Constructor have same name as class name.	6) Destructor have same name as class name but with **tlide (~)** sign.

7) Syntax:
Class_name(arguments)
{
//Body of the constructor
}
	7) Syntax:
	~Class_name()
	{
	//Body of the destructor
	}

12. **Discuss the various situations when a copy constructor is automatically invoked. How a default constructor can be equivalent to constructor having default arguments.**

Various situations when the copy constructor is called or automatically invoked are:

- When compiler generates the temporary object.
- When an object is constructed based on the object of the same class.
- When an object of the class is passed to function by values as an argument.
- When an object of the class is returned by value.

Constructor with default arguments is a is parameterized constructor which has default values in arguments. Thus the arguments defined in such constructor are optional. We may or may not pass arguments while defining object of the class. When an object is created with no supplied values, the default values are used. By this way, constructor with default constructor is equivalent to default constructor.

Example:

```
#include<iostream>
using namespace std;
class Box
{
private:
float l,b,h;
public:
Box(float le=10,float br=5,float he=5)
{
l=le;
b=br;
h=he;
}
void displaymember()
{
cout<<"Length,breadth and height"<<l<<b<<h<<endl;
}
float getvolume()
{
return (l*b*h);
}
};
int main()
{
Box b;
float vol;
vol=b.getvolume();
cout<<"Volume="<<vol<<endl;
return 0;
}
```

13. What is de-constructor? can you have two destructors in a class? Give example to support your reason.[PU:2014 spring]

De-constructor is a member function that destroys the object that have been created by a constructor.
Destructor doesn't take any arguments, which means destructor cannot be overloaded. So, that there will be only one destructor in a class.
Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed. A destructor is called for a class object when that object passes out of scope or is explicitly deleted.
So, that there cannot be more than one destructor (two destructor) in a same class.
(Write example of destructor yourself)

14. Differentiate methods of argument passing in constructor and destructor.

A constructor is allowed to accept the arguments as the arguments can be used to initialize the data members of the class.
A destructor does not accept any arguments as its only work is to deallocate the memory of the object.
In class, there can be multiple constructors which are identified by the number arguments passed.
In class, there is only one destructor.
Constructors can be overload to perform different action under the name of the same constructor whereas, destructors cannot be overloaded.

Example:

```
#include <iostream>
using namespace std;
class test
{
private:
int a;
public:
test()
{
a=10;
cout<<"Default constructor is called"<<endl;
}
test(int x)
{
a=x;
cout<<"Parameterized constructor is called"<<endl;
}
~test()
{
cout<<"Destructor is called"<<endl;
}
void display()
{
cout<<"value of a="<<a<<endl;
}
};
```

```
int main()
{
test t1;
test t2(5);
t1.display();
t2.display();
return 0;
}
```

15. What do you mean by stack vs Heap? Explain memory recovery. Explain the use of new and delete operator. [PU:2009 spring]

Stack

- It's a region of computer's memory that stores temporary variables created by each function (including the main() function).
- The stack is a "Last in First Out" data structure and limited in size. Every time a function declares a new variable, it is "pushed" (inserted) onto the stack.
- Every time a function exits, all of the variables pushed onto the stack by that function, are freed or popped (that is to say, they are deleted).
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

Heap

- The heap is a region of computer's memory that is not managed automatically and is not as tightly managed by the CPU.
- We must allocate and de-allocate variable explicitly. (for allocating variable new and for freeing variables delete operator is used.
- It does not have a specific limit on memory size.

Memory recovery

Because in most languages objects are dynamically allocated, they must be recovered at run-time. There are two broad approaches to this:

- Force the programmer to explicitly say when a value is no longer being used: delete aCard; // C++ example
- Use a garbage collection system that will automatically determine when values are no longer being used, and recover the memory.

Use of new and delete operator

new is used to allocate memory blocks at run time (dynamically). While, delete is used to de-allocate the memory which has been allocated dynamically.

Example of new and delete in C+

```
#include <iostream>
using namespace std;
int main ()
{
int i,n;
int *ptr;
int sum=0;
cout << "How many numbers would you like to Enter? ";
cin >>n;
ptr= new int[n];
for (i=0; i<n; i++)
{
cout << "Enter number:"<<i+1<<endl;
cin >> ptr[i];
}
for (i=0; i<n; i++)
{
sum=sum+ptr[i];
}
cout<<"sum of numbers="<<sum<<endl;
delete[] ptr;
return 0;
}
```

16. What are the advantages of dynamic memory allocation? Explain with suitable example. [PU:2016 spring]

The main advantage of using dynamic memory allocation is preventing the wastage of memory or efficient utilization of memory. Let us consider an example:
In static memory allocation the memory is allocated before the execution of program begins (During compilation).In this type of allocation the memory cannot be resized after initial allocation. So it has some limitations. Like

- Wastage of memory
- Overflow of memory

eg. int num[100];
Here, the size of an array has been fixed to 100.If we just enter to 10 elements only, then their will be wastage of 90 memory location and if we need to store more than 100 elements there will be memory overflow.
But, in dynamic memory allocation memory is allocated during runtime.so it helps us to allocate and de-allocate memory during the period of program execution as per requirement. So there will be proper utilization of memory.
In C++ dynamic memory allocation can be achieved by using new and delete operator. **new** is used to allocate memory blocks at run time (dynamically). While, **delete** is used to de-allocate the memory which has been allocated dynamically.

Let us consider an example:

```
#include <iostream>
using namespace std;
int main ()
{
int i,n;
int *ptr;
int sum=0;
cout << "How many numbers would you like to Enter? ";
cin >>n;
ptr= new int[n];
for (i=0; i<n; i++)
{
cout << "Enter number:"<<i+1<<endl;
cin >> ptr[i];
}
for (i=0; i<n; i++)
{
sum=sum+ptr[i];
}
cout<<"sum of numbers="<<sum<<endl;
delete[] ptr;
return 0;
}
```

Here, in above example performs the sum of n numbers .But the value of n will be provided during runtime.so that memory allocation to store n numbers is done during program execution which results proper utilization of memory and preventing from wastage of memory or overflow of memory.

17.Constructor is called in derived class but it can't be inherited. Support your answer with suitable example.

In object-oriented programming, a constructor is a special method that is called when an object is created. The constructor initializes the object's data members and sets the object's initial state. In C++, constructors are not inherited by derived classes. However, the derived class can call the constructor of the base class to initialize the base class members.

Here is an example that demonstrates this concept:

```
#include <iostream>
using namespace std;
class Base
{
private:
int a;
public:
   Base(int x)
   {
                    a=x;
      cout << "Base constructor called "<<endl;
   }
   void showbase()
   {
         cout<<"value initialzied in base constructor="<<a<<endl;
         }
};

class Derived : public Base
{
private:
int b;
public:
   Derived(int x,int y) : Base(x)
         {
      b=y;
      cout << "Derived constructor called"<<endl;
   }
   void showderived()
   {
         cout<<"value initialized in derived constructor="<<b<<endl;
         }
};
```

```
int main()
 {
   Derived d(5,10);
   d.showbase();
   d.showderived();
   return 0;
}
```

Output:

```
Base constructor called
Derived constructor called
value initialized in base constructor=5
value initialized in derived constructor=10
```

In this example, we have a base class 'Base' and a derived class 'Derived'. The Base class has a constructor that takes one integer parameter, and the Derived class has a constructor that takes two integer parameter and calls the Base constructor by passing one parameter.

When we create an object of the Derived class in the main function and pass the integer value of 5 and 10 to its constructor i.e. Derived d(5,10).

Now, derived class called the base class constructor by supplying first value .i.e 5 is initialized to base class data member 'a' ,followed by the constructor of the Derived class constructor executed and initialized the derived class data member with value 10.

Thus, even though the Derived class does not inherit the constructor of the Base class, it can still call the constructor of the Base class explicitly in its own constructor to initialize the base class members.