**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **24/06/2020** | **Name:** | **Namratha S Hipparagi** |
| **Course:** | **C++** | **USN:** | **4AL16EC040** |
| **Topic:** | **Classes and objects**  **More on classes** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **namrathahipparagi\_1** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **MODULE 5**  A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object. For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.    **ABSTARCTION**  Data Abstraction in C++ Advertisements. Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.  **Encapsulation**  Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.  for example, a capsule which is mixed of several medicines. ... Now we can use setter and getter methods to set and get the data in it. The Java Bean class is the example of a fully encapsulated class.  **Constructors**.  A constructor in C++ is a special method that is automatically called when an object of a class is created. When a class or struct is created, its constructor is called. Constructors have the same name as the class or struct, and they usually initialize the data members of the new object. In the following example, a class named Taxi is defined by using a simple constructor. ... For more information, see Instance Constructors.  **MODULE 6**   **Separate files for classes** C++ classes (and often function prototypes) are normally split up into two files. The header file has the extension of. h and contains class definitions and functions. The implementation of the class goes.  **Destructors**  A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete. A destructor has the same name as the class, preceded by a tilde ( ~ ). For example, the destructor for class String is declared: ~String() .  Its syntax is same as constructor except the fact that it is preceded by the tilde sign.  ~class\_name() { }; //syntax of destructor  **Selection of operator**  C++Syntax: Member selection: -> .  This syntax is used to select a data member or a member function given a pointer to an object (->) or an object (.) Selecting a data member means getting direct access to a members data, at most objects won't allow that as it breaks the hiding (or encapsulation) ofitsdata.  **Member initializer**  Constructor is a special non-static [member function](https://en.cppreference.com/w/cpp/language/member_functions) of a class that is used to initialize objects of its class type. In the definition of a constructor of a class, *member initializer list* specifies the initializers for direct and virtual bases and non-static data members. ( Not to be confused with [std::initializer list](https://en.cppreference.com/w/cpp/utility/initializer_list) ) **Syntax** Constructors are declared using member [function declarators](https://en.cppreference.com/w/cpp/language/function) of the following form:   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | | | | | | | | | | | *class-name* **(** *parameter-list*(optional) **)** *except-spec*(optional) *attr*(optional) |  |  |  |  |  |  |  |  |  |   **Operator overloading**  In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.  A simple and complete example  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | #include<iostream>  using namespace std;    class Complex {  private:      int real, imag;  public:      Complex(int r = 0, int i =0)  {real = r;   imag = i;}        // This is automatically called when '+' is used with      // between two Complex objects      Complex operator + (Complex const &obj) {           Complex res;           res.real = real + obj.real;           res.imag = imag + obj.imag;           return res;      }      void print() { cout << real << " + i" << imag << endl; }  };    int main()  {      Complex c1(10, 5), c2(2, 4);      Complex c3 = c1 + c2; // An example call to "operator+"      c3.print();  } |   Output:  12 + i9 |