**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | 24-6-2020 | **Name:** | Rasika Patil |
| **Course:** | **C++** | **USN:** | 4AL16EC057 |
| **Topic:** | * Classes and objects * More on classes | **Semester & Section:** | 8th B |
| **Github Repository:** | Rasika B Patil |  |  |

|  |  |  |
| --- | --- | --- |
| **MODULE 5**  **CLASSES AND OBJECTS** C:\Users\User\Downloads\WhatsApp Image 2020-06-24 at 8.28.42 PM.jpeg C:\Users\User\Downloads\WhatsApp Image 2020-06-24 at 8.28.42 PM (3).jpegC++ Classes and Objects **Class:** 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.   * 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 defines the properties and behavior of the objects in a Class. * In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.   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.  Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.  Attributes and methods are basically variables and functions that belongs to the class. These are often referred to as "class members".  A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects. Create a Class To create a class, use the class keyword: Example Create a class called "MyClass":  class MyClass {       // The class   public:             // Access specifier     int myNum;        // Attribute (int variable)     string myString;  // Attribute (string variable) }; Create an Object In C++, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.  To create an object of MyClass, specify the class name, followed by the object name.  To access the class attributes (myNum and myString), use the dot syntax (.) on the object: Example Create an object called "myObj" and access the attributes:  class MyClass {       // The class   public:             // Access specifier     int myNum;        // Attribute (int variable)     string myString;  // Attribute (string variable) };  int main() {   MyClass **myObj**;  // Create an object of MyClass    // Access attributes and set values   **myObj.**myNum = 15;    **myObj.**myString = "Some text";    // Print attribute values   cout << myObj.myNum << "\n";   cout << myObj.myString;   return 0; } Abstraction in C++ Data abstraction is one of the most essential and important feature of object oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.  Consider a real life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.  **Abstraction using Classes:** We can implement Abstraction in C++ using classes. Class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to outside world and which is not. Encapsulation in C++ In normal terms **Encapsulation**is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them. Consider a real life example of encapsulation, in a company there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keep records of all the data related to finance. Similarly the sales section handles all the sales related activities and keep records of all the sales. Now there may arise a situation when for some reason an official from finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of sales section and the employees that can manipulate them are wrapped under a single name “sales section”. Constructors in C++ **What is constructor?** A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.  **How constructors are different from a normal member function?**  A constructor is different from normal functions in following ways:   * Constructor has same name as the class itself * Constructors don’t have return type * A constructor is automatically called when an object is created. * If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).   **MODULE 6**  **MORE ON CLASSES**  **C:\Users\User\Downloads\WhatsApp Image 2020-06-24 at 8.28.42 PM (1).jpeg** Destructors in C++ **What is destructor?** Destructor is a member function which destructs or deletes an object.  **When is destructor called?** A destructor function is called automatically when the object goes out of scope: (1) the function ends (2) the program ends (3) a block containing local variables ends (4) a delete operator is called  **How destructors are different from a normal member function?** Destructors have same name as the class preceded by a tilde (~) Destructors don’t take any argument and don’t return anything  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | class String  {  private:      char \*s;      int size;  public:      String(char \*); // constructor      ~String();      // destructor  };    String::String(char \*c)  {      size = strlen(c);      s = new char[size+1];      strcpy(s,c);  }    String::~String()  {      delete []s;  } |  Const member functions in C++ Like member functions and member function arguments, the objects of a class can also be declared as **const**. an object declared as const cannot be modified and hence, can invoke only const member functions as these functions ensure not to modify the object. A const object can be created by prefixing the const keyword to the object declaration. Any attempt to change the data member of const objects results in a compile-time error. **Syntax:**  const Class\_Name Object\_name;   * When a function is declared as const, it can be called on any type of object, const object as well as non-const objects. * Whenever an object is declared as const, it needs to be initialized at the time of declaration. however, the object initialization while declaring is possible only with the help of constructors.   A function becomes const when the const keyword is used in the function’s declaration. The idea of const functions is not to allow them to modify the object on which they are called. It is recommended the practice to make as many functions const as possible so that accidental changes to objects are avoided.  **Object composition**  In real-life, complex objects are often built from smaller, simpler objects. For example, a car is built using a metal frame, an engine, some tires, a transmission, a steering wheel, and a large number of other parts. A personal computer is built from a CPU, a motherboard, some memory, etc… Even you are built from smaller parts: you have a head, a body, some legs, arms, and so on. This process of building complex objects from simpler ones is called **object composition**.  Broadly speaking, object composition models a “has-a” relationship between two objects. A car “has-a” transmission. Your computer “has-a” CPU. You “have-a” heart. The complex object is sometimes called the whole, or the parent. The simpler object is often called the part, child, or component.  In C++, you’ve already seen that structs and classes can have data members of various types (such as fundamental types or other classes). When we build classes with data members, we’re essentially constructing a complex object from simpler parts, which is object composition. For this reason, structs and classes are sometimes referred to as **composite types**.  Object Composition is useful in a C++ context because it allows us to create complex classes by combining simpler, more easily manageable parts. This reduces complexity, and allows us to write code faster and with less errors because we can reuse code that has already been written, tested, and verified as working.  **Types of object composition**  There are two basic subtypes of object composition: composition and aggregation. We’ll examine composition in this lesson, and aggregation in the next.  A note on terminology: the term “composition” is often used to refer to both composition and aggregation, not just to the composition subtype. In this tutorial, we’ll use the term “object composition” when we’re referring to both, and “composition” when we’re referring specifically to the composition subtype.  **Composition**  To qualify as a **composition**, an object and a part must have the following relationship:   * The part (member) is part of the object (class) * The part (member) can only belong to one object (class) at a time * The part (member) has its existence managed by the object (class) * The part (member) does not know about the existence of the object (class)   A good real-life example of a composition is the relationship between a person’s body and a heart. Let’s examine these in more detail.  Composition relationships are part-whole relationships where the part must constitute part of the whole object. For example, a heart is a part of a person’s body. The part in a composition can only be part of one object at a time. A heart that is part of one person’s body can not be part of someone else’s body at the same time.  In a composition relationship, the object is responsible for the existence of the parts. Most often, this means the part is created when the object is created, and destroyed when the object is destroyed. But more broadly, it means the object manages the part’s lifetime in such a way that the user of the object does not need to get involved. For example, when a body is created, the heart is created too. When a person’s body is destroyed, their heart is destroyed too. Because of this, composition is sometimes called a “death relationship”. Operator Overloading in C++ 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();  } | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |