**DAILY ASSESSMENT FORMAT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **25/06/2020** | **Name:** | **Abhishek Vasudev Mahendrakar** | |
| **Course:** | **C++ Tutorial by SOLOLEARN** | **USN:** | **4AL17EC003** | |
| **Topic:** | 1. **Inheritance and Polymorphism** 2. **Templates, Exceptions and Files** | **Semester & Section:** | **6th-‘A’** | |
| **Github Repository:** | **ECEAbhishekVMahendrakar** | **E-mail:** | **abhi2244mahendrakar@gmail.com** | |
| **FORENOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Report – Report can be typed or hand written for up to two pages.**   1. **Inheritance and Polymorphism:** 2. **Inheritance:**  * **Inheritance**is one of the most important concepts of object-oriented programming. Inheritance allows us to define a class based on another class. This facilitates greater ease in creating and maintaining an application. * The class whose properties are inherited by another class is called the **Base** class. * The class which inherits the properties is called the **Derived** class. * The Base class is specified using a **colon**and an **access specifier**:**public**means, that all public members of the base class are public in the derived class.  1. **Protected Members:**  * Public members may be accessed from anywhere outside of the class, while access to private members is limited to their class and friend functions. * There is one more access specifier - **protected**. A **protected**member variable or function is very similar to a private member, with one difference - it can be accessed in the derived classes. * **Public**Inheritance: public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class. * **Protected**Inheritance: public and protected members of the base class become protected members of the derived class. * **Private**Inheritance: public and protected members of the base class become private members of the derived class.  1. **Derived Class, Constructor and Destructor:**  * When the object is destroyed, the derived class's destructor is called, and then the base class' destructor is called. * **Constructors** The base class constructor is called first. * **Destructors** The derived class destructor is called first, and then the base class destructor gets called.  1. **Polymorphism:**  * Polymorphism occurs when there is a hierarchy of classes and they are related by **inheritance**. * C++ polymorphism means that a call to a member function will cause a **different**implementation to be executed depending on the **type**of object that invokes the function.  1. **Virtual functions:**  * A virtual function in the base class, with a corresponding version in a derived class, allows polymorphism to use Enemy pointers to call the derived classes' functions. * If a function in the base class is **virtual**, the function's implementation in the derived class is called according to the actual type of the object referred to, regardless of the declared type of the pointer.  1. **Abstract Classes:**  * The virtual member functions without definition are known as **pure virtual functions**. They basically specify that the derived classes define that function on their own. * A **pure virtual function**basically defines, that the derived classes will have that function defined on their own. * Every derived class inheriting from a class with a pure virtual function **must** override that function. * You **cannot**create objects of the base class with a pure virtual function. * These classes are called **abstract**. They are classes that can only be used as base classes, and thus are allowed to have pure virtual functions.  1. **Templates, Exceptions and Files:** 2. **Function Templates:**  * With function templates, the basic idea is to avoid the necessity of specifying an exact type for each variable. Instead, C++ provides us with the capability of defining functions using placeholder types, called **template type parameters**. * To define a function template, use the keyword **template**, followed by the template type definition. * When creating a template type parameter, the keyword **typename**may be used as an alternative to the keyword **class**: **template <typename T>.** * Template functions can save a lot of time, because they are written only once, and work with different types. * Template functions reduce code maintenance, because duplicate code is reduced significantly.  1. **Function Templates with multiple parameters:**  * Function templates also make it possible to work with **multiple**generic data types. Define the data types using a comma-separated list. * **T** is short for Type, and is a widely used name for type parameters.  1. **Class Templates:**  * We can also define **class templates**, allowing classes to have members that use template parameters as types.  1. **Template Specialization:**  * **Template specialization** allows for the definition of a different implementation of a template when a specific type is passed as a template argument. * We precede the class name with **template<>**, including an empty parameter list. This is because all types are known and no template arguments are required for this specialization, but still, it is the specialization of a class template, and thus it requires to be noted as such. * It is important that this prefix, is the <**char**> specialization parameter after the class template name. This specialization parameter itself identifies the type for which the template class is being specialized (**char**).  1. **Exceptions:**  * Problems that occur during program execution are called **exceptions**. * In C++ exceptions are responses to anomalies that arise while the program is running, such as an attempt to divide by zero. * C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**. * **throw** is used to throw an exception when a problem shows up. * A **try**block identifies a block of code that will activate specific exceptions. It's followed by one or more **catch**blocks. The **catch**keyword represents a block of code that executes when a particular exception is thrown.  1. **More on Exceptions:**  * Exception handling is particularly useful when dealing with user input. * we catch exceptions of type **integer**only. It's possible to specify that your catch block handles any type of exception thrown in a try block. To accomplish this, add an **ellipsis (...)**between the parentheses of catch  1. **Working with files:**  * Another useful C++ feature is the ability to read and write to files. That requires the standard C++ library called **fstream**. * Three new data types are defined in fstream: **ofstream**: Output file stream that creates and writes information to files. **ifstream**: Input file stream that reads information from files. **fstream**: General file stream, with both ofstream and ifstream capabilities that allow it to create, read, and write information to files. * To perform file processing in C++, header files <**iostream**> and <**fstream**> must be included in the C++ source file. * A file must be opened before you can read from it or write to it. * The **open()**function to open the "test.txt" file on the file system. * Close it using the member function **close()**.  1. **More on Files:**  * Provide the path to your file using the **ofstream**objects constructor, instead of calling the **open** function. * Under certain circumstances, such as when you don't have file permissions, the **open**function can fail. * The **is\_open()** member function checks whether the file is open and ready to be accessed. * An optional second parameter of the **open**function defines the **mode**in which the file is opened.      * All these flags can be combined using the bitwise operator OR (|). | | | |