**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **25/06/2020** | **Name:** | **Lepakshi T V** |
| **Course:** | **C++** | **USN:** | **4AL17EC044** |
| **Topic:** | * **Inheritance and polymorphism** * **Templates, exceptions and files** | **Semester & Section:** | **6th sem A sec** |
| **Github Repository:** | **Lepakshi-044** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report – Report can be typed or hand written for up to two pages.**  **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. For example, the **Daughter**class (derived) can be inherited from the **Mother**class (base). The derived class inherits all feature from the base class, and can have its own additional features. A derived class inherits all base class methods with the following exceptions.  Constructors, destructors. Overloaded operators. The friend functions.  **Access Specifiers:**  Up to this point, we have worked exclusively with **public**and **private**access specifiers. Public members may be accessed from anywhere outside of the class, while access to private members is limited to their class and friend functions.  **Protected:** 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.  **Constructors** The base class constructor is called first.  **Destructors** The derived class destructor is called first, and then the base class destructor gets called.  **Polymorphism**  The word **polymorphism**means "having many forms". Typically, polymorphism occurs when there is a hierarchy of classes and they are related by **inheritance**.  **Virtual Functions**  The previous example demonstrates the use of base class pointers to the derived classes. Why is that useful? Continuing on with our game example, we want every Enemy to have an **attack ()** function. To be able to call the corresponding attack () function for each of the derived classes using Enemy pointers, we need to declare the base class function as **virtual**. Defining 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.  **Pure Virtual Function**  In some situations, you'd want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class. 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. The syntax is to replace their definition by =0 (an equal sign and a zero). Abstract Classes They are classes that can only be used as base classes, and thus are allowed to have pure virtual functions.  **Function Templates**  Functions and classes help to make programs easier to write, safer, and more maintainable. However, while functions and classes do have all of those advantages, in certain cases they can also be somewhat limited by C++'s requirement that you specify types for all of your parameters. 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 templatetypeparameters.  Function templates also make it possible to work with **multiple**generic data types. Define the data types using a comma-separated list. Let's create a function that compares arguments of varying data types (an **int**and a **double**), and prints the smaller one. **T** is short for Type, and is a widely used name for type parameters. It's not necessary to use **T**, however; you can declare your type parameters using any identifiers that work for you.  **Class Templates:** A specific syntax is required in case you define your member functions outside of your class for example in a separate source file. You need to specify the **generic type** in angle brackets after the class name.  **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.  **Throwing Exceptions:** C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**. **throw** is used to throw an exception when a problem shows up. Catching Exceptions: 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. Code that could generate an exception is surrounded with the **try/catch**block. You can specify what type of exception you want to catch by the exception declaration that appears in parentheses following the keyword **catch**.  **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.  **Opening a File:** A file must be opened before you can read from it or write to it. Either the **ofstream**or **fstream**object may be used to open a file for writing.  **Closing a File:** When you've finished working with a file, close it using the member function **close ()**.  Attended webinar on VLSI |