**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **25/06/2020** | **Name:** | **Lavanya B** |
| **Course:** | **C++ programming** | **USN:** | **4al17ec043** |
| **Topic:** | **Inheritance**  **Polymorphism** | **Semester & Section:** | **6th A** |
| **Github Repository:** | **Lavanya-B** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **This keyword**  **Every object in C++ has access to its own address through an important pointer called the this pointer.**  **Inside a member function this may be used to refer to the invoking object.**  **The this keyword has an important role in operator overloading.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class MyClass {**  **public:**  **MyClass(int a) : var(a)**  **{ }**  **void printInfo() {**  **cout << var <<endl;**  **cout << this->var <<endl;**  **cout << (\*this).var <<endl;**  **}**  **private:**  **int var;**  **};**  **int main() {**  **MyClass obj(42);**  **obj.printInfo();**  **}**  **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).**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class Mother**  **{**  **public:**  **Mother() {};**  **void sayHi() {**  **cout << "Hi";**  **}**  **};**  **class Daughter: public Mother**  **{**  **public:**  **Daughter() {};**  **};**  **int main() {**  **Daughter d;**  **d.sayHi();**  **}**  **Polymorphism**  **Polymorphism means "having many forms".**  **Typically, 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.**  **Suppose you want to make a simple game, which includes different enemies: monsters, ninjas, etc. All enemies have one function in common: an attack function. However, they each attack in a different way. In this situation, polymorphism allows for calling the same attack function on different objects, but resulting in different behaviors.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class Enemy {**  **protected:**  **int attackPower;**  **public:**  **void setAttackPower(int a){**  **attackPower = a;**  **}**  **};**  **class Ninja: public Enemy {**  **public:**  **void attack() {**  **cout << "Ninja! - "<<attackPower<<endl;**  **}**  **};**  **class Monster: public Enemy {**  **public:**  **void attack() {**  **cout << "Monster! - "<<attackPower<<endl;**  **}**  **};**  **int main() {**  **Ninja n;**  **Monster m;**  **Enemy \*e1 = &n;**  **Enemy \*e2 = &m;**  **e1->setAttackPower(20);**  **e2->setAttackPower(80);**  **n.attack();**  **m.attack();**  **}**  **Virtual function**  **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.**  **Eg:**  **#include <iostream>**  **using namespace std;**  **class Enemy {**  **public:**  **virtual void attack() { }**  **};**  **class Ninja: public Enemy {**  **public:**  **void attack() {**  **cout << "Ninja!"<<endl;**  **}**  **};**  **class Monster: public Enemy {**  **public:**  **void attack() {**  **cout << "Monster!"<<endl;**  **}**  **};**  **int main() {**  **Ninja n;**  **Monster m;**  **Enemy \*e1 = &n;**  **Enemy \*e2 = &m;**  **e1->attack();**  **e2->attack();**  **}** |