**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. Can a virtual function be a friend of another class?
   1. Yes
   2. **No**
2. A virtual function is redefined in which of the following class?
   1. **Derived class**
   2. Parent class
   3. base class
   4. Both A and B
3. Which type of binding is determined by the type of the object pointed to or referenced, not by the type of the pointer or reference?
   1. Early binding
   2. Late binding
   3. **Virtual binding**
   4. Static binding
4. In C++, how can you achieve early binding for a function that is not virtual?
   1. By using the 'final' keyword
   2. **By using the 'static' keyword**
   3. By using the 'const' keyword
   4. Early binding is not possible for non-virtual functions
5. In C++, what is the purpose of the 'typeid' operator?
   1. To determine the type of an object at compile-time
   2. **To determine the type of an object at runtime**
   3. To convert between different types
   4. To get the memory address of an object
6. How Exception handling is implemented in the C++ program?
   1. Using Exception keyword
   2. **Using try-catch block**
   3. Using Exception block
   4. Using Error handling schedules
7. Which of the following is an exception in C++?
   1. **Divide by zero**
   2. Semicolon not written
   3. Variable not declared
   4. An expression is wrongly written
8. What is the purpose of std::exception in C++?
   1. To handle exceptions automatically
   2. **To serve as a base class for user-defined exceptions**
   3. To terminate the program on exception
   4. To specify exceptions in the code
9. How do you specify template arguments explicitly when calling a template function in C++?
   1. By using the "template" keyword followed by the template arguments
   2. By enclosing the template arguments in square brackets [ ]
   3. **By providing the template arguments in angle brackets < >**
   4. By using the "template" keyword followed by the function name and arguments
10. What is the purpose of using the "typename" keyword when defining a member function of a class template outside the class definition?
    1. To indicate that the function is a member function
    2. To specify the return type of the member function
    3. **To indicate that the following identifier is a template parameter**
    4. To disambiguate between member function and non-member function templates

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

1. What is the output of the following code?

#include <iostream>

class Base {

public:

virtual void display() const {

std::cout << "Base ";

}

};

class Derived : public Base {

public:

void display() const override {

std::cout << "Derived ";

}

};

int main() {

const Base\* b = new Base();

const Derived\* d = new Derived();

b->display();

d->display();

return 0;

}

* 1. Base Base
  2. Derived Derived
  3. **Base Derived**
  4. Derived Base

1. What is the output of the following code?

#include <iostream>

class Vehicle {

public:

virtual void honk() {

std::cout << "Vehicle honk" << std::endl;

}

};

class Car : public Vehicle {

public:

void honk() override {

std::cout << "Car honk" << std::endl;

}

};

void sound(Vehicle& v) {

v.honk();

}

int main() {

Car c;

sound(c);

return 0;

}

* 1. Vehicle honk
  2. **Car honk**
  3. Compiler error
  4. Runtime error

1. What is the output of this program?

#include <iostream>

using namespace std;

int main()

{

int P = -1;

try {

cout << "Inside try";

if (P < 0)

{

throw P;

cout << "After throw";

}

}

catch (int P ) {

cout << " Exception Caught";

}

cout << " After catch";

return 0;

}

* 1. Inside try Exception Caught After throw After catch
  2. **Inside try Exception Caught After catch**
  3. Inside try Exception Caught
  4. Inside try After throw After catch

1. Which of the following statements are true about Catch handler?

i) It must be placed immediately after try block T.

ii) It can have multiple parameters.

iii) There must be only one catch handler for every try block.

iv) There can be multiple catch handler for a try block T.

v) Generic catch handler can be placed anywhere after try block.

* 1. Only i, iv, v
  2. Only i, ii, iii
  3. **Only i, iv**
  4. Only i, ii

1. What will be the output of this program?

#include <iostream>

using namespace std;

template<int n> struct funStruct

{

static const int val = 2\*funStruct<n-1>::val;

};

template<> struct funStruct<0>

{

static const int val = 1 ;

};

int main()

{

cout << funStruct<10>::val << endl;

return 0;

}

* 1. Compiler Error
  2. 1
  3. 2
  4. **1024**

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Create an abstract class Character with pure virtual functions attack() and defend().

Derive classes Warrior and Mage from Character and implement the attack() and defend() functions in each.

Use these classes to create a simple game character system.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Character\* player1 = new Warrior;  player1->attack(); | Character\* player1 = new Warrior;  player1->defend(); | Character\* player2 = new Mage;  player2->attack(); |
| **Output** | Warrior attacks with a sword | Warrior defends with a shield | Mage attacks with a fireball |

Solution :

**#include <iostream>**

**using namespace std;**

**class Character {**

**public:**

**// Pure virtual functions 'attack()' and 'defend()'.**

**// These functions are meant to be implemented by derived classes to provide specific attack and defend actions for each character.**

**virtual void attack() = 0;**

**virtual void defend() = 0;**

**};**

**class Warrior : public Character {**

**public:**

**// Override of the 'attack()' function for the 'Warrior' class.**

**// Provides the specific attack implementation for warriors: "Warrior attacks with a sword."**

**void attack() override {**

**cout << "Warrior attacks with a sword." << endl;**

**}**

**// Override of the 'defend()' function for the 'Warrior' class.**

**// Provides the specific defend implementation for warriors: "Warrior defends with a shield."**

**void defend() override {**

**cout << "Warrior defends with a shield." << endl;**

**}**

**};**

**class Mage : public Character {**

**public:**

**// Override of the 'attack()' function for the 'Mage' class.**

**// Provides the specific attack implementation for mages: "Mage attacks with a fireball."**

**void attack() override {**

**cout << "Mage attacks with a fireball." << endl;**

**}**

**// Override of the 'defend()' function for the 'Mage' class.**

**// Provides the specific defend implementation for mages: "Mage defends with a magic barrier."**

**void defend() override {**

**cout << "Mage defends with a magic barrier." << endl;**

**}**

**};**

**int main() {**

**// Creating objects of 'Warrior' and 'Mage' using base class pointers 'Character\*'.**

**Character\* player1 = new Warrior;**

**Character\* player2 = new Mage;**

**// Calling the 'attack()' and 'defend()' functions for each character using the base class pointers.**

**// This demonstrates polymorphism, as the correct version of each function is called based on the actual object type at runtime.**

**player1->attack(); // Warrior attacks with a sword**

**player1->defend(); // Warrior defends with a shield**

**player2->attack(); // Mage attacks with a fireball**

**player2->defend(); // Mage defends with a magic barrier**

**// Cleaning up the dynamically allocated objects.**

**delete player1;**

**delete player2;**

**return 0;**

**}**

Q17) Create a class Person with attributes name and age. You are tasked to perform exception handling in class constructors if age is not in range 1-120.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Person person("Alice", 150); | Person person("Alice", 100); | Person person("Alice", -50); |
| **Output** | Error: Invalid age value! | Name: Alice, Age: 100 | Error: Invalid age value! |

Solution :

**#include <iostream>**

**#include <string>**

**// Class representing a Person with a name and age**

**class Person {**

**private:**

**std::string name;**

**int age;**

**public:**

**// Constructor to initialize a Person with a name and age**

**Person(const std::string& n, int a) : name(n) {**

**if (a < 0 || a > 120) {**

**// Check if the age value is within the valid range (0 to 120)**

**// If the age is invalid, throw a runtime\_error with an error message**

**throw std::runtime\_error("Invalid age value!");**

**}**

**age = a;**

**}**

**// Function to display the name and age of the Person**

**void display() const {**

**std::cout << "Name: " << name << ", Age: " << age << std::endl;**

**}**

**};**

**int main() {**

**try {**

**// Try to create a Person object with name "Alice" and age 150**

**// This should throw an "Invalid age value!" exception**

**Person person("Alice", 150);**

**person.display(); // This line will not be executed due to the exception**

**}**

**catch (const std::exception& ex) {**

**// Catch any exception that derives from std::exception (including std::runtime\_error)**

**// Print the error message associated with the caught exception**

**std::cout << "Error: " << ex.what() << std::endl;**

**}**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) You are tasked to create a flexible function to concatenate various data types based on two or more parameters.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | x = 5 y = 10 | x = 3.14 y = 2.71 | x=Hello, y=World! z=Welcome |
| **Output** | Output: Sum of integers: 15 | Sum of doubles: 5.85 | Concatenated String with three strings: Hello, world!Welcome |

Solution :

**#include <iostream>**

**using namespace std;**

**// Function template to add two values of the same data type.**

**template <typename T>**

**T add(T a, T b) {**

**return a + b;**

**}**

**// Function template overloaded to concatenate two strings.**

**template <typename T>**

**T add(T a, T b, T c) {**

**return a + b + c;**

**}**

**int main() {**

**int x = 5, y = 10;**

**cout << "Sum of integers: " << add(x, y) << endl; // Output: Sum of integers: 15**

**double a = 3.14, b = 2.71;**

**cout << "Sum of doubles: " << add(a, b) << endl; // Output: Sum of doubles: 5.85**

**string str1 = "Hello, ", str2 = "world!";**

**cout << "Concatenated String: " << add(str1, str2) << endl; // Output: Concatenated String: Hello, world!**

**string str3 = "Welcome ";**

**cout << "Concatenated String with three strings: " << add(str1, str2, str3) << endl;**

**// Output: Concatenated String with three strings: Hello, world!Welcome**

**return 0;**

**}**