**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. The virtual function is used to tell the compiler to perform \_\_\_\_\_\_\_\_?
   1. static linkage
   2. dynamic linkage
   3. late binding
   4. **Both B and C**
2. Which of the following best describes the virtual function?
   1. Function overriding
   2. write a function in the child class that is already present in the parent class
   3. Run time polymorphism
   4. **All of these**
3. Which keyword is used to invoke the base class version of an overridden virtual function?
   1. base
   2. super
   3. parent
   4. **Using scope resolution operator (::)**
4. If a class inherits an abstract class and does not provide definitions for all pure virtual functions, it becomes:
   1. Concrete class
   2. **Abstract class**
   3. Static class
   4. Interface class
5. Which of the following statements about function overriding is true?
   1. **The access specifier for the overridden function in the derived class must be the same as the base class**
   2. Overridden functions cannot have the same name as the base class function
   3. Overridden functions cannot have different parameters from the base class function
   4. Overridden functions cannot have a different return type from the base class function
6. By default, what a program does when it detects an exception?
   1. Continue running
   2. **Results in the termination of the program**
   3. Calls other functions of the program
   4. Removes the exception and tells the programmer about an exception
7. Why do we need to handle exceptions?
   1. **To avoid unexpected behavior of a program during run-time**
   2. To let compiler remove all exceptions by itself
   3. To successfully compile the program
   4. To get correct output
8. In C++, can a function that does not throw any exceptions be called inside a try block?
   1. **Yes**
   2. No
9. What is the purpose of using the "noexcept" specifier in C++?
   1. **To specify that a function will not throw any exceptions**
   2. To specify that a function will always throw exceptions
   3. To indicate that a function can only throw specific exceptions
   4. To indicate that a function can throw any type of exception
10. What is the function resolution order when calling a function with the same name but different types in C++?
    1. **The non-template function is preferred over the template function.**
    2. The template function is preferred over the non-template function.
    3. The compiler will randomly choose one of the functions.
    4. The program will not compile due to ambiguity.

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

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

1. What is the output of the following code?

#include <iostream>

class A {

public:

virtual void foo() {

std::cout << "A::foo()" << std::endl;

}

};

class B : public A {

public:

void foo() override {

std::cout << "B::foo()" << std::endl;

}

};

int main() {

A\* a\_ptr = new B;

B\* b\_ptr = dynamic\_cast<B\*>(a\_ptr);

if (b\_ptr) {

b\_ptr->foo();

} else {

std::cout << "Dynamic cast failed" << std::endl;

}

return 0;

}

* 1. A::foo()
  2. **B::foo()**
  3. Compiler error
  4. Dynamic cast failed

1. What is the output of the following code?

try {

throw 42;

} catch (int num) {

std::cout << "Caught an integer: " << num << std::endl;

} catch (...) {

std::cout << "Caught an unknown exception." << std::endl;

}

* 1. **Caught an integer: 42**
  2. Caught an unknown exception.
  3. Error: Unhandled exception of type int.
  4. The program will not compile.

1. What is the output of this program?

#include <iostream>

using namespace std;

int main()

{

try

{

throw 10;

}

catch (...)

{

cout << "Default Exceptionn";

}

catch (int param)

{

cout << "Int Exceptionn";

}

return 0;

}

* 1. Default Exception
  2. Int Exception
  3. **Compiler Error**
  4. None of the above

1. What happens when this C++ program is compiled?

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

class A

{

int a;

public:

A(){}

};

class B: public A

{

int b;

public:

B(){}

};

void func()

{

B b;

throw b;

}

int main()

{

try{

func();

}

catch(B \*b){

cout<<"Caught B Class\n";

}

catch(A a){

cout<<"Caught A Class\n";

}

}

* 1. Caught B Class
  2. **Caught A Class**
  3. Compile-time error
  4. Run-time error

1. What will be the output of this program?

#include <iostream>

using namespace std;

template <int i>

void fun()

{

i = 20;

cout << i;

}

int main()

{

fun<10>();

return 0;

}

* 1. **Compile Error**
  2. 10
  3. 20
  4. 15

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

Q16) Create an abstract class Drawable with pure virtual functions draw() and resize().

Derive classes Circle and Rectangle from Drawable and implement the draw() and resize() functions in each.

Use these classes to simulate drawing and resizing shapes.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Drawable\* shape1 = new Circle;  shape1->draw(); | Drawable\* shape1 = new Circle;  shape1->resize(); | Drawable\* shape2 = new Rectangle;  shape2->draw(); |
| **Output** | Drawing a circle. | Resizing the circle | Drawing a rectangle |

Solution :

**#include <iostream>**

**using namespace std;**

**class Drawable {**

**public:**

**// Pure virtual functions 'draw()' and 'resize()'.**

**// These functions are meant to be implemented by derived classes to provide specific draw and resize actions for each shape.**

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

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

**};**

**class Circle : public Drawable {**

**public:**

**// Override of the 'draw()' function for the 'Circle' class.**

**// Provides the specific draw implementation for circles: "Drawing a circle."**

**void draw() override {**

**cout << "Drawing a circle." << endl;**

**}**

**// Override of the 'resize()' function for the 'Circle' class.**

**// Provides the specific resize implementation for circles: "Resizing the circle."**

**void resize() override {**

**cout << "Resizing the circle." << endl;**

**}**

**};**

**class Rectangle : public Drawable {**

**public:**

**// Override of the 'draw()' function for the 'Rectangle' class.**

**// Provides the specific draw implementation for rectangles: "Drawing a rectangle."**

**void draw() override {**

**cout << "Drawing a rectangle." << endl;**

**}**

**// Override of the 'resize()' function for the 'Rectangle' class.**

**// Provides the specific resize implementation for rectangles: "Resizing the rectangle."**

**void resize() override {**

**cout << "Resizing the rectangle." << endl;**

**}**

**};**

**int main() {**

**// Creating objects of 'Circle' and 'Rectangle' using base class pointers 'Drawable\*'.**

**Drawable\* shape1 = new Circle;**

**Drawable\* shape2 = new Rectangle;**

**// Calling the 'draw()' and 'resize()' functions for each shape 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.**

**shape1->draw(); // Draws a circle**

**shape1->resize(); // Resizes the circle**

**shape2->draw(); // Draws a rectangle**

**shape2->resize(); // Resizes the rectangle**

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

**delete shape1;**

**delete shape2;**

**return 0;**

**}**

Q17) Simulate a bank account. Allow deposits and withdrawals, but throw exceptions for invalid balance and withdrawal amounts.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | BankAccount account("123456789", 1000);  account.withdraw(1500); | BankAccount account("123456789", 900);  account.withdraw(300); | BankAccount account("123456789", 1000);  account.deposit(-500); |
| **Output** | Error: Insufficient balance! | Withdrawal successful! | Error: Invalid deposit amount! |

Solution :

**#include <iostream>**

**#include <string>**

**class BankAccount {**

**private:**

**std::string accountNumber;**

**double balance;**

**public:**

**BankAccount(std::string accNum, double initialBalance) : accountNumber(accNum), balance(initialBalance) {}**

**// Function to deposit money into the account**

**void deposit(double amount) {**

**if (amount <= 0)**

**throw "Invalid deposit amount!";**

**balance += amount;**

**}**

**// Function to withdraw money from the account**

**void withdraw(double amount) {**

**if (amount <= 0)**

**throw "Invalid withdrawal amount!";**

**if (amount > balance)**

**throw "Insufficient balance!";**

**balance -= amount;**

**std::cout << "Withdrawal successful! ";**

**}**

**// Function to get the current balance of the account**

**double getBalance() const {**

**return balance;**

**}**

**};**

**int main() {**

**BankAccount account("123456789", 1000);**

**try {**

**// Perform some transactions**

**account.deposit(500); // Deposit $500**

**account.withdraw(300); // Withdraw $300**

**account.withdraw(1500); // Should throw "Insufficient balance!"**

**// account.deposit(-500); // Should throw "Invalid deposit amount!"**

**}**

**catch (const char\* error) {**

**std::cout << "Error: " << error << std::endl;**

**}**

**return 0;**

**}**

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

Q18) You are a talented software engineer working on an exciting project to develop a playlist reordering system for a popular music streaming platform. As part of this endeavor, you have been tasked with creating a function template to efficiently reverse the order of songs in a playlist, allowing users to enjoy their favorite tracks in a different sequence.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {1, 2, 3, 4, 5} | {1.1, 2.2, 3.3, 4.4} | {‘a’,’b’,’c’,’d’} |
| **Output** | Reversed array: 5 4 3 2 1 | Reversed array: 4.4 3.3 2.2 1.1 | Reversed array: d c b a |

Solution :

**#include <iostream>**

**using namespace std;**

**// Function template 'reverseArray' reverses the order of elements in an array.**

**// The template takes two parameters: 'arr', representing the array of type 'T',**

**// and 'N', the size of the array.**

**// The function efficiently reverses the array by swapping corresponding elements from both ends.**

**template <typename T, size\_t N>**

**void reverseArray(T arr[N]) {**

**for (size\_t i = 0; i < N / 2; i++) {**

**T temp = arr[i];**

**arr[i] = arr[N - 1 - i];**

**arr[N - 1 - i] = temp;**

**}**

**}**

**int main() {**

**// Example 1: Reverse an array of integers.**

**int intArr[] = {1, 2, 3, 4, 5};**

**reverseArray<int, 5>(intArr);**

**cout << "Reversed array: ";**

**for (int i = 0; i < 5; i++) {**

**cout << intArr[i] << " "; // Output: Reversed array: 5 4 3 2 1**

**}**

**cout << endl;**

**// Example 2: Reverse an array of doubles.**

**double doubleArr[] = {1.1, 2.2, 3.3, 4.4};**

**reverseArray<double, 4>(doubleArr);**

**cout << "Reversed array: ";**

**for (int i = 0; i < 4; i++) {**

**cout << doubleArr[i] << " "; // Output: Reversed array: 4.4 3.3 2.2 1.1**

**}**

**cout << endl;**

**return 0;**

**}**