**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. Early binding can be implemented by which of the following?
   1. Virtual function
   2. Function overriding
   3. **Function overloading**
   4. B and C
2. Which statement is true about virtual functions?
   1. Virtual functions can be static members
   2. Virtual functions cannot be static members
   3. Virtual functions must be members of some class
   4. **B and C**
3. Can a virtual function be defined in the base class and overridden in the derived class?
   1. **Yes**
   2. No
   3. Only if the base class is abstract
   4. Only if the derived class is abstract
4. When using pointers or references, which binding occurs automatically for virtual functions?
   1. Early binding
   2. **Late binding**
   3. Both early and late binding
   4. None of the above
5. If a derived class does not provide its own destructor, what happens when the object is deleted?
   1. The program will throw a compilation error
   2. **The base class destructor will be called automatically**
   3. The program will crash at runtime
   4. The compiler will provide a default destructor
6. What is the role of the "std::exception::virtual const char\* what() const noexcept" function?
   1. It throws an exception.
   2. It catches an exception.
   3. **It returns the error message associated with the exception.**
   4. It handles uncaught exceptions.
7. What is an exception in C++ program?
   1. **A problem that arises during the execution of a program**
   2. A problem that arises during compilation
   3. Also known as the syntax error
   4. Also known as semantic error
8. What is the purpose of using the "nothrow" keyword with "new" in C++?
   1. **To suppress the exception if the memory allocation fails**
   2. To allocate memory on the stack instead of the heap
   3. To allocate memory without initializing it
   4. To allocate memory with a specific alignment
9. Can we catch multiple exceptions using a single catch block in C++?
   1. **Yes**
   2. No
10. What happens when you overload a function template and a non-template function with the same name and arguments in C++?
    1. The program will not compile due to ambiguity.
    2. **The non-template function is preferred over the template function.**
    3. The template function is preferred over the non-template function.
    4. The program will compile, but the behavior is undefined.

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

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

1. Which of the following is true about the following program

#include <iostream>

using namespace std;

template <class P, class Q, class R>

class A {

P x;

Q y;

R z;

static int count;

};

int main()

{

A<int, int, int> m;

A<char, char, char> n;

cout << sizeof(m) << endl;

cout << sizeof(n) << endl;

return 0;

}

* 1. Compiler Error: template parameters cannot have default values
  2. 12 6
  3. **12 3**
  4. 6 3

1. What is the output of the following code?

#include <iostream>

class Shape {

public:

virtual void draw() const {

std::cout << "Drawing shape" << std::endl;

}

};

class Circle : public Shape {

public:

void draw() const override {

std::cout << "Drawing circle" << std::endl;

}

};

int main() {

Circle c;

const Shape\* s\_ptr = &c;

s\_ptr->draw();

return 0;

}

* 1. Drawing shape
  2. **Drawing circle**
  3. Compiler error
  4. Runtime error

1. What is the output of the following code?

try {

throw std::runtime\_error("An error occurred.");

} catch (std::exception& ex) {

std::cout << "Caught an exception: " << ex.what() << std::endl;

} catch (...) {

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

}

* 1. **Caught an exception: An error occurred.**
  2. Caught an unknown exception.
  3. Error: Unhandled exception of type std::runtime\_error.
  4. The program will not compile.

1. What is the output of this program?

#include <iostream>

using namespace std;

int main()

{

try

{

throw 'b';

}

catch (int param)

{

cout << "Int Exception";

}

catch (...)

{

cout << "Default Exception";

}

cout << "After Exception";

return 0;

}

* 1. **Default Exception After Exception**
  2. Int Exception After Exception
  3. Int Exception
  4. Default Exception

1. What is the output of the following code?

#include <iostream>

int main() {

try {

throw 10;

}

catch (int x) {

std::cout << "Caught an integer: " << x;

}

catch (...) {

std::cout << "Caught something else.";

}

return 0;

}

* 1. **Caught an integer: 10**
  2. Caught something else.
  3. The program will terminate.
  4. Compiler error

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

Q16) Create an abstract class Shape with a pure virtual function area(). Add a protected data member 'name' to the Shape class. Derive classes Circle and Square from Shape and implement the area() function in each. Use these classes to demonstrate accessing the protected data member from derived classes.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Shape\* shape1 = new Circle(5);  shape1->area(); | Shape\* shape2 = new Square(4);  shape2->area(); | Shape\* shape1 = new Circle(13);  shape1->area(); |
| **Output** | Area of Circle:78.5397 | Area of Square:16 | Area of Circle: 530.929 |

Solution :

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Shape {**

**protected:**

**string name; // Protected data member to store the name of the shape**

**public:**

**// Constructor to initialize the 'name' of the shape**

**Shape(const string& n) : name(n) {}**

**// Pure virtual function 'area()'.**

**// This function is meant to be implemented by derived classes to calculate the area of their specific shape.**

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

**};**

**class Circle : public Shape {**

**private:**

**float radius;**

**public:**

**Circle(float r) : Shape("Circle"), radius(r) {}**

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

**// Calculates and prints the area of the circle using the formula: pi \* r^2**

**void area() override {**

**cout << "Area of " << name << ": " << 3.14159f \* radius \* radius << endl;**

**}**

**};**

**class Square : public Shape {**

**private:**

**float side;**

**public:**

**Square(float s) : Shape("Square"), side(s) {}**

**// Override of the 'area()' function for the 'Square' class.**

**// Calculates and prints the area of the square using the formula: side \* side**

**void area() override {**

**cout << "Area of " << name << ": " << side \* side << endl;**

**}**

**};**

**int main() {**

**// Creating objects of 'Circle' and 'Square' through their base class pointer 'Shape\*'.**

**Shape\* shape1 = new Circle(13);**

**Shape\* shape2 = new Square(4);**

**// Using the 'area()' function of the 'Circle' and 'Square' classes through the base class pointer.**

**// This demonstrates polymorphism, as the correct version of 'area()' is called based on the actual object type at runtime.**

**shape1->area(); // Prints the area of the circle**

**shape2->area(); // Prints the area of the square**

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

**delete shape1;**

**delete shape2;**

**return 0;**

**}**

Q17) You are developing a scientific calculator application that performs various mathematical operations, including division. Your task is to implement division functionality while handling the case when the user attempts to divide a number by zero by catching exceptions at different levels.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter two numbers: 6 2 | Enter two numbers: 7 0 | Enter two numbers: 0 0 |
| **Output** | Result: 3 | Inner Exception: Division by zero! | Inner Exception: Division by zero! |

Solution :

**#include <iostream>**

**int divideNumbers(int dividend, int divisor) {**

**if (divisor == 0) {**

**throw std::runtime\_error("Division by zero!");**

**}**

**return dividend / divisor;**

**}**

**int main() {**

**try {**

**int num1, num2;**

**std::cout << "Enter two numbers: ";**

**std::cin >> num1 >> num2;**

**try {**

**int result = divideNumbers(num1, num2);**

**std::cout << "Result: " << result << std::endl;**

**}**

**catch (const std::runtime\_error& ex) {**

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

**}**

**}**

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

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

**}**

**return 0;**

**}**

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

Q18) You are an innovative software engineer working on a challenging project to create an automated sorting system for a busy warehouse.

As part of this critical endeavor, you have been assigned the task of designing a function template to find the minimum element in an array of products based on their numerical attributes.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {3.0, 4.5, 1.2, 2.7} | {10, 5, 8, 3, 15} | {-10, -5, -8, -3, -15} |
| **Output** | Min: 1.2 | Min: 3 | Min: -15 |

Solution :

**#include <iostream>**

**using namespace std;**

**// Function template 'findMin' finds the minimum element in an array.**

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

**// The function iterates through the array to identify the minimum value using a comparison algorithm.**

**// It then returns the minimum value found in the array.**

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

**T findMin(T arr[N]) {**

**T minVal = arr[0];**

**for (size\_t i = 1; i < N; i++) {**

**if (arr[i] < minVal) {**

**minVal = arr[i];**

**}**

**}**

**return minVal;**

**}**

**int main() {**

**// Create an array of integers and find the minimum value.**

**int intArr[] = {10, 5, 8, 3, 15};**

**cout << "Min: " << findMin<int, 5>(intArr) << endl; // Output: Min: 3**

**// Create an array of doubles and find the minimum value.**

**double doubleArr[] = {3.0, 4.5, 1.2, 2.7};**

**cout << "Min: " << findMin<double, 4>(doubleArr) << endl; // Output: Min: 1.2**

**return 0;**

**}**