**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. What is a virtual base class in C++?
   1. A base class whose objects cannot be created directly.
   2. A base class that can be inherited multiple times in a class hierarchy.
   3. A base class that contains virtual functions.
   4. **A base class that is declared as "virtual."**
2. What is the purpose of a virtual function in C++?
   1. To allow multiple inheritance in classes.
   2. To improve performance in class hierarchies.
   3. **To allow a derived class to provide its implementation.**
   4. To prevent the creation of objects of the base class.
3. What is the other name for early binding?
   1. Static binding
   2. Dynamic binding
   3. Runtime binding
   4. **Compile-time binding**
4. What is the other name for late binding?
   1. Static binding
   2. **Dynamic binding**
   3. Runtime binding
   4. Compile-time binding
5. What is the purpose of a virtual destructor in C++?
   1. To deallocate memory allocated using new
   2. **To allow derived class destructors to be called through a base class pointer**
   3. To enable multiple inheritance
   4. To prevent memory leaks
6. Which keyword is used to declare a virtual function in C++?
   1. **virtual**
   2. polymorphic
   3. abstract
   4. dynamic
7. What is exception handling in C++?
   1. **A mechanism to handle errors and abnormal situations during program execution.**
   2. A way to optimize program performance.
   3. A method to print debug messages.
   4. A feature to avoid using pointers.
8. Which standard C++ header file should be included for using exception handling features?
   1. **<exception>**
   2. <error>
   3. <trycatch>
   4. <stdexcept>
9. What symbol is used to specify a template parameter in C++?
   1. \*
   2. $
   3. #
   4. **typename or class**
10. What is the role of the "std::exception::what()" 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.

**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() {

std::cout << "Base Display ";

}

};

class Derived : public Base {

public:

void display() override {

std::cout << "Derived Display ";

}

};

int main() {

Base\* b = new Derived();

b->display();

return 0;

}

* 1. Base Display
  2. **Derived Display**
  3. Compiler Error
  4. Undefined Behavior

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
5. What is the output of the following code?

#include <iostream>

class Base {

public:

virtual ~Base() { std::cout << "Base destructor\n"; }

};

class Derived : public Base {

public:

~Derived() { std::cout << "Derived destructor\n"; }

};

int main() {

Base\* ptr = new Derived;

delete ptr;

return 0;

}

1. Base destructor
2. Derived destructor
3. **Both Base and Derived destructors**
4. Neither Base nor Derived destructor
5. 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.
5. What will be the output of the following code?

#include <iostream>

void throw\_exception() {

throw "An exception occurred!";

}

int main() {

try {

throw\_exception();

}

catch (const char\* ex) {

std::cout << ex;

}

return 0;

}

1. exception!
2. **An exception occurred!**
3. The program will terminate.
4. Compiler error

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

Q16) Develop a graphical user interface (GUI) library with a virtual base class Widget and a derived class Button.

Implement a virtual function draw() in the base class and override it in the Button class to display the button on the screen.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Widget\* widget = new Button(100, 50, "Click Me!");  widget->draw(); | Widget\* widget2 = new Widget(140, 250);  widget2->draw(); | Widget\* widget = new Button(200, 250, "Subscribe!");  widget->draw(); |
| **Output** | Drawing a button with label "Click Me!" at (100, 50) | Drawing a generic widget at (140, 250) | Drawing a button with label "Subscribe!" at (200, 250) |

Solution :

**#include <iostream>**

**#include <string>**

**using namespace std;**

**// Base class representing a Widget**

**class Widget {**

**protected:**

**int x, y; // Coordinates of the widget on the screen**

**public:**

**Widget(int \_x, int \_y) : x(\_x), y(\_y) {}**

**// Virtual function to draw the widget**

**virtual void draw() const {**

**cout << "Drawing a generic widget at (" << x << ", " << y << ")" << endl;**

**}**

**};**

**// Derived class representing a Button, inheriting from Widget**

**class Button : public Widget {**

**private:**

**string label; // Label of the button**

**public:**

**Button(int \_x, int \_y, string \_label) : Widget(\_x, \_y), label(\_label) {}**

**// Override the draw function to display the button**

**void draw() const override {**

**cout << "Drawing a button with label \"" << label << "\" at (" << x << ", " << y << ")" << endl;**

**}**

**};**

**int main() {**

**Widget\* widget = new Button(100, 50, "Click Me!");**

**// When we call the draw method on the base class pointer, the virtual function**

**// will be dispatched to the appropriate version based on the actual object type.**

**widget->draw();**

**Widget\* widget2 = new Widget(140, 250);**

**widget2->draw(); // Output: Drawing a generic widget at (140, 250)**

**delete widget;**

**return 0;**

**}**

Q17) Write a C++ program that creates a string and attempts to access an out-of-range element.

Handle the length\_error exception.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | str = "Hello, world!"  Index- 9 | str = "Hello, world!";  index- 13 | str = "Hello, world!";  Index-21 |
| **Output** | Character at index 9: r | Exception: Index out of range! | Exception: Index out of range! |

Solution :

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**string str = "Hello, world!";**

**int index;**

**cout << "Enter an index to access the character: ";**

**cin >> index;**

**try {**

**// Check if the index is valid (within the string's bounds)**

**if (index < 0 || index >= str.length()) {**

**// If the index is invalid, throw a 'length\_error' exception with a corresponding message**

**throw length\_error("Index out of range!");**

**}**

**// If the index is valid, print the character at the specified index**

**cout << "Character at index " << index << ": " << str[index] << endl;**

**} catch (const length\_error& e) {**

**// Catch the 'length\_error' exception if the index is out of range**

**cout << "Exception: " << e.what() << endl;**

**}**

**return 0;**

**}**

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

Q18) You are designing a math library to handle complex numbers in a variety of applications.

Implement a template function complexAddSubtract that takes two complex numbers as input and performs addition and subtraction operations.

The function should be able to handle complex numbers with different data types. Test the function with various scenarios involving different data types for complex numbers.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Integer Complex Numbers:  Number 1: 3 + 2i  Number 2: 1 + 5i | Floating-Point Complex Numbers:  Number 3: 2.5 + 3.1i  Number 4: 1.2 + 4i | Integer Complex Numbers:  Number 1: 5 + 1i  Number 2: 3 + 5i |
| **Output** | Addition: 4 + 7i  Subtraction: 2 - 3i | Addition: 3.7 + 7.1i  Subtraction: 1.3 - 0.9i | Addition: 8 + 6i  Subtraction: 2 - 4i |

Solution :

**#include <iostream>**

**using namespace std;**

**// Complex number class template**

**template <typename T>**

**class Complex {**

**private:**

**T real;**

**T imaginary;**

**public:**

**Complex(T real, T imaginary) : real(real), imaginary(imaginary) {}**

**// Addition operator overload**

**Complex<T> operator+(const Complex<T>& other) const {**

**return Complex<T>(real + other.real, imaginary + other.imaginary);**

**}**

**// Subtraction operator overload**

**Complex<T> operator-(const Complex<T>& other) const {**

**return Complex<T>(real - other.real, imaginary - other.imaginary);**

**}**

**// Display the complex number**

**void display() const {**

**cout << real;**

**if (imaginary >= 0) {**

**cout << " + " << imaginary << "i";**

**} else {**

**cout << " - " << -imaginary << "i";**

**}**

**cout << endl;**

**}**

**};**

**int main() {**

**// Test with integers**

**Complex<int> num1(3, 2);**

**Complex<int> num2(1, 5);**

**cout << "Integer Complex Numbers:" << endl;**

**cout << "Number 1: ";**

**num1.display(); // Output: Number 1: 3 + 2i**

**cout << "Number 2: ";**

**num2.display(); // Output: Number 2: 1 + 5i**

**cout << "Addition: ";**

**(num1 + num2).display(); // Output: Addition: 4 + 7i**

**cout << "Subtraction: ";**

**(num1 - num2).display(); // Output: Subtraction: 2 - 3i**

**// Test with floating-point numbers**

**Complex<float> num3(2.5f, 3.1f);**

**Complex<float> num4(1.2f, 4.0f);**

**cout << "\nFloating-Point Complex Numbers:" << endl;**

**cout << "Number 3: ";**

**num3.display(); // Output: Number 3: 2.5 + 3.1i**

**cout << "Number 4: ";**

**num4.display(); // Output: Number 4: 1.2 + 4i**

**cout << "Addition: ";**

**(num3 + num4).display(); // Output: Addition: 3.7 + 7.1i**

**cout << "Subtraction: ";**

**(num3 - num4).display(); // Output: Subtraction: 1.3 - 0.9i**

**return 0;**

**}**