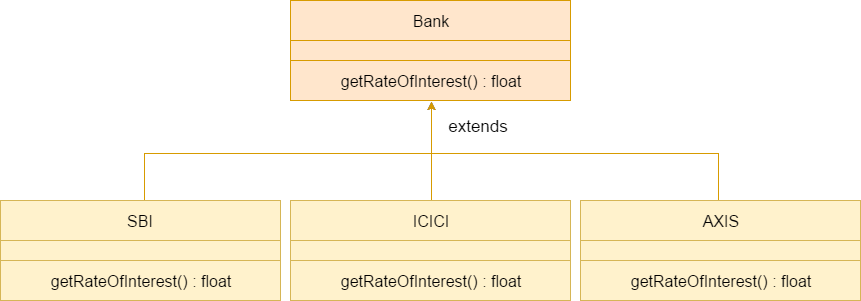
**Assignment 5**

(1). WRITE DOWN THE CODE FOR FOLLOWING DIAGRAM

USING POLYMORPHISM



Program:

(1). WRITE DOWN SOURCE FOR THE EXCEPTIONS FOR

FOLLOWING SNIPPETS,

• Separation of Error Handling code from Normal

Code

• Functions/Methods can handle any exceptions

• Notes on Catch all

• If an exception is thrown and not caught anywhere,

the program terminates abnormally

**Program:**

#include <iostream>

using namespace std;

class Bank {

public:

virtual float getRateOfInterest() = 0;

};

class AXIS : public Bank {

public:

float getRateOfInterest() {

return 8.5f;

}

};

class ICICI : public AXIS {

public:

float getRateOfInterest() {

return 10.0f;

}

};

class SBI : public ICICI {

public:

float getRateOfInterest() {

return 12.5f;

}

};

int main() {

Bank \*b1 = new AXIS();

Bank \*b2 = new ICICI();

Bank \*b3 = new SBI();

cout << "Rate of Interest for AXIS: " << b1->getRateOfInterest() << endl;

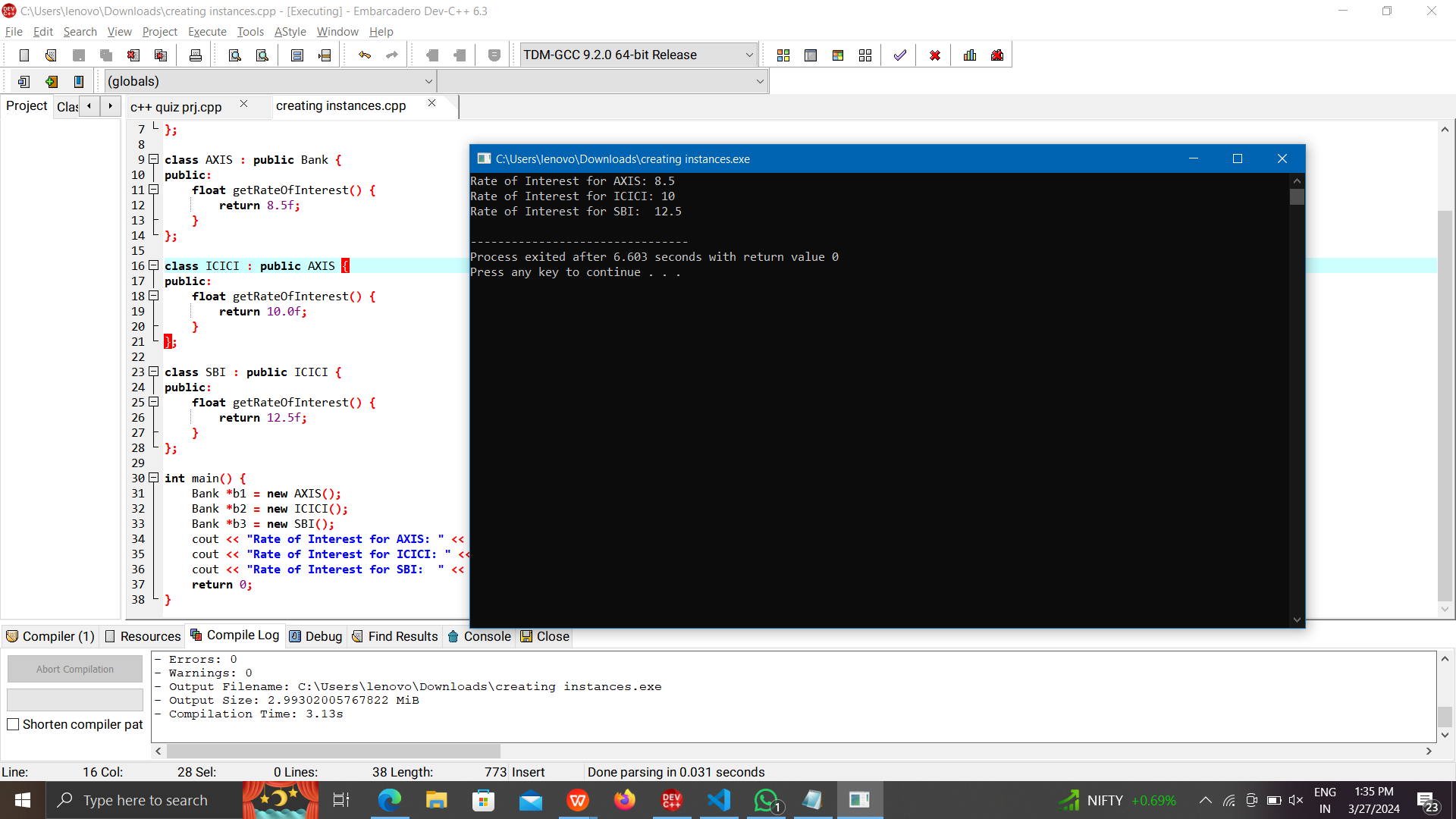
cout << "Rate of Interest for ICICI: " << b2->getRateOfInterest() << endl;

cout << "Rate of Interest for SBI: " << b3->getRateOfInterest() << endl;

return 0;

}

**Output:**



(2). WRITE DOWN SOURCE FOR THE EXCEPTIONS FOR

FOLLOWING SNIPPETS,

• Separation of Error Handling code from Normal

Code

• Functions/Methods can handle any exceptions

• Notes on Catch all

• If an exception is thrown and not caught anywhere,

the program terminates abnormally

**Program:**

#include <iostream>

#include <exception>

class CustomException : public std::exception {

public:

const char\* what() const throw() {

return "Custom Exception occurred!";

}

};

void simulateException(bool shouldThrow) {

if (shouldThrow) {

throw CustomException();

}

}

int main() {

try {

std::cout << "Executing normal code..." << std::endl;

simulateException(true);

std::cout << "This line will not be printed." << std::endl;

} catch (CustomException& e) {

std::cerr << "Caught Custom Exception: " << e.what() << std::endl;

} catch (std::exception& e) {

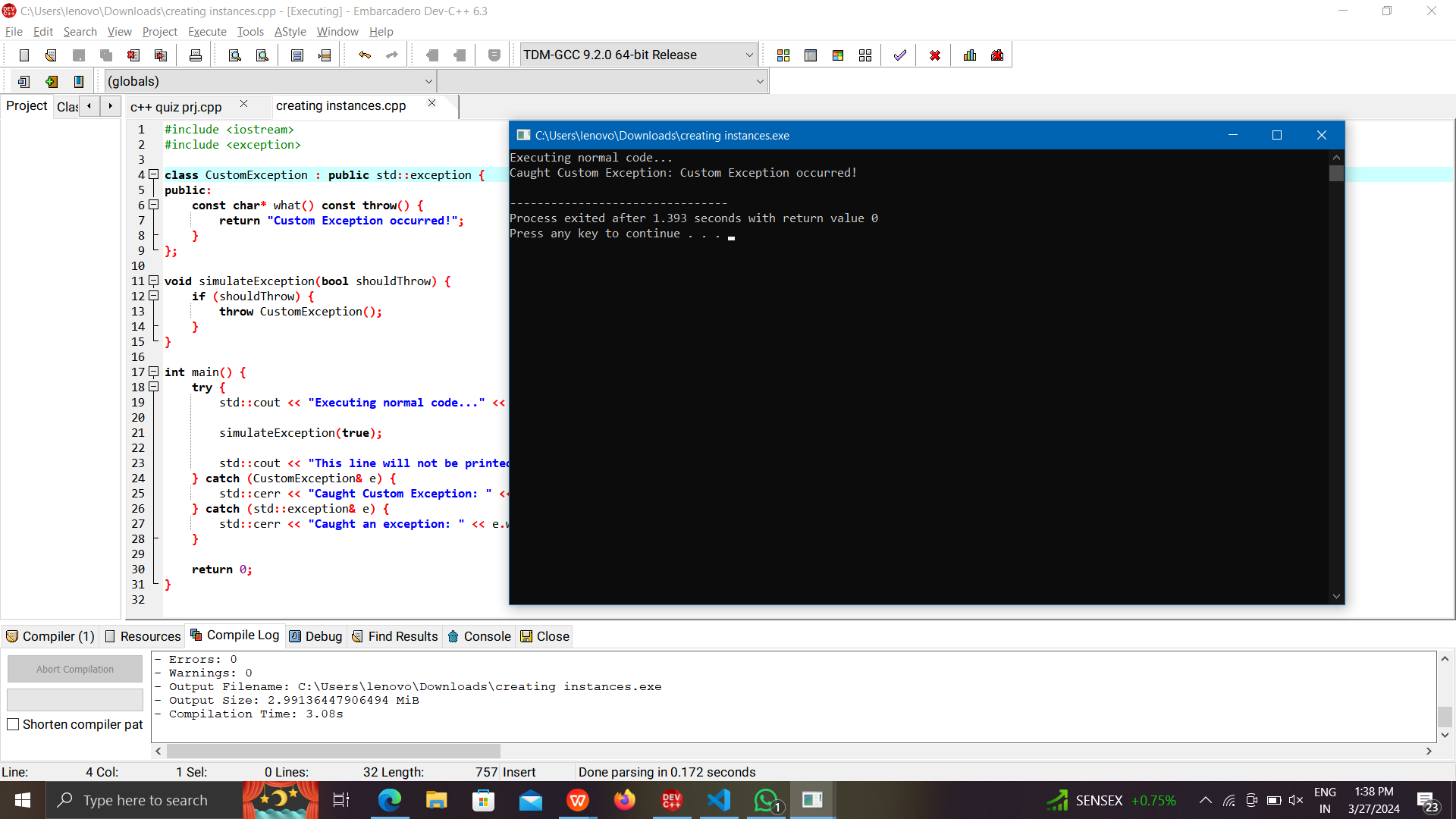
std::cerr << "Caught an exception: " << e.what() << std::endl;

}

return 0;

}

**Output:**



(3). Write down the C++ for Students Report with the implementation of

Virtual Function.

**Program:**

#include <iostream>

#include <string>

class Student {

protected:

std::string name;

int rollNumber;

public:

Student(const std::string& n, int roll) : name(n), rollNumber(roll) {}

virtual void displayInfo() {

std::cout << "Name: " << name << std::endl;

std::cout << "Roll Number: " << rollNumber << std::endl;

}

virtual int calculateTotalMarks() = 0;

};

class ScienceStudent : public Student {

protected:

int physicsMarks;

int chemistryMarks;

int mathMarks;

public:

ScienceStudent(const std::string& n, int roll, int phy, int chem, int math)

: Student(n, roll), physicsMarks(phy), chemistryMarks(chem), mathMarks(math) {}

void displayInfo() override {

Student::displayInfo();

std::cout << "Physics Marks: " << physicsMarks << std::endl;

std::cout << "Chemistry Marks: " << chemistryMarks << std::endl;

std::cout << "Math Marks: " << mathMarks << std::endl;

}

int calculateTotalMarks() override {

return physicsMarks + chemistryMarks + mathMarks;

}

};

class ArtsStudent : public Student {

protected:

int historyMarks;

int geographyMarks;

int literatureMarks;

public:

ArtsStudent(const std::string& n, int roll, int history, int geo, int lit)

: Student(n, roll), historyMarks(history), geographyMarks(geo), literatureMarks(lit) {}

void displayInfo() override {

Student::displayInfo();

std::cout << "History Marks: " << historyMarks << std::endl;

std::cout << "Geography Marks: " << geographyMarks << std::endl;

std::cout << "Literature Marks: " << literatureMarks << std::endl;

}

int calculateTotalMarks() override {

return historyMarks + geographyMarks + literatureMarks;

}

};

int main() {

ScienceStudent scienceStud("Alice", 101, 85, 90, 88);

ArtsStudent artsStud("Bob", 102, 78, 85, 92);

std::cout << "Science Student Information:" << std::endl;

scienceStud.displayInfo();

std::cout << "Total Marks: " << scienceStud.calculateTotalMarks() << std::endl;

std::cout << "\nArts Student Information:" << std::endl;

artsStud.displayInfo();

std::cout << "Total Marks: " << artsStud.calculateTotalMarks() << std::endl;

return 0;

}

**Output:**

