ASSIGNMENT 6:

Name: Om Prasanna Kulkarni

PRN: 22510034

Batch: S3

Branch: CSE

Topic: Basics Of Object Oriented Programming

1. EXCEPTION HANDLING Write a program to perform arithmetic operations on any given two numbers which throws an exception if the dividend is zero or does not contain an operator. Use of multiple try-catch statements is mandatory to perform this assignment. Provide specific ERROR names for the errors occurring during the exception handling.  
  
SOURCE CODE:

#include <iostream>

using namespace std;

int main() {

    double num1, num2;

    char op;

    try {

        cout << "Enter two numbers: ";

        cin >> num1 >> num2;

        cout << "Enter an operator (+, -, \*, /): ";

        cin >> op;

        try {

            if (op == '+') {

                cout << "Result: " << num1 + num2 << endl;

            } else if (op == '-') {

                cout << "Result: " << num1 - num2 << endl;

            } else if (op == '\*') {

                cout << "Result: " << num1 \* num2 << endl;

            } else if (op == '/') {

                if (num2 == 0) {

                    throw "Division by zero error";

                }

                cout << "Result: " << num1 / num2 << endl;

            } else {

                throw "Invalid operator error";

            }

        } catch (const char \*error) {

            cerr << "Error: " << error << endl;

        }

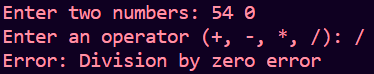
    } catch (ios\_base::failure &e) {

        cerr << "Error: Invalid input. Please enter valid numbers." << endl;

    }

    return 0;

}

OUTPUT:  


2. EXCEPTION HANDLING Write a program to accept an email address and throw an exception if the given string does not contain '@' symbol in it To perform this assignment, use of exception and its keywords is mandatory. Use of multiple try-catch statements is mandatory to perform this assignment. Provide specific ERROR names for the errors occurring during the exception handling.

SOURCE CODE:  
#include <iostream>

#include <string>

using namespace std;

class EmailFormatException : public exception {

public:

    const char\* what() const noexcept {

        return "Error: Invalid email format. Missing '@' symbol.";

    }

};

int main() {

    string email;

    try {

        cout << "Enter an email address: ";

        cin >> email;

        try {

            if (email.find('@') == string::npos) {

                throw EmailFormatException();

            }

            cout << "Email address: " << email << endl;

        } catch (const EmailFormatException &e) {

            cerr << e.what() << endl;

        }

    } catch (ios\_base::failure &e) {

        cerr << "Error: Invalid input. Please enter a valid email address." << endl;

    }

    return 0;

}

OUTPUT:  


3. TEMPLATE Write a program for a function template for finding the minimum value contained in an array. Expected Output: The minimum value of array (6,8,3,1,9) is = 1 The minimum value of array (1.3,1.5,0.5,0.200,2.5) is = 0.200.

SOURCE CODE:  
#include <iostream>

template <typename T>

T findMinimum(const T array[], int size) {

    T min = array[0];

    for (int i = 1; i < size; ++i) {

        if (array[i] < min) {

            min = array[i];

        }

    }

    return min;

}

int main() {

    int intArray[] = {6, 8, 3, 1, 9};

    double doubleArray[] = {1.3, 1.5, 0.5, 0.200, 2.5};

    int intArraySize = sizeof(intArray) / sizeof(intArray[0]);

    int doubleArraySize = sizeof(doubleArray) / sizeof(doubleArray[0]);

    int minInt = findMinimum(intArray, intArraySize);

    double minDouble = findMinimum(doubleArray, doubleArraySize);

    std::cout << "The minimum value of integer array is = " << minInt << std::endl;

    std::cout << "The minimum value of double array is = " << minDouble << std::endl;

    return 0;

}

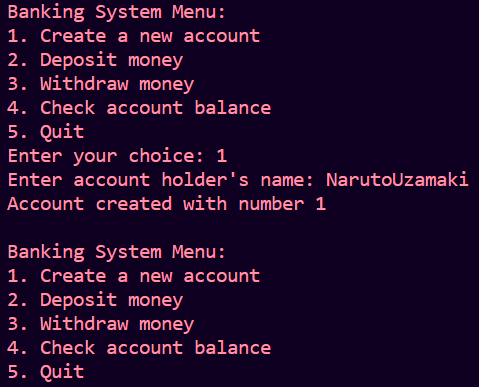
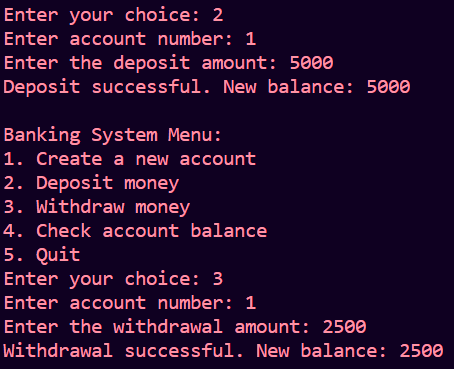
OUTPUT:  

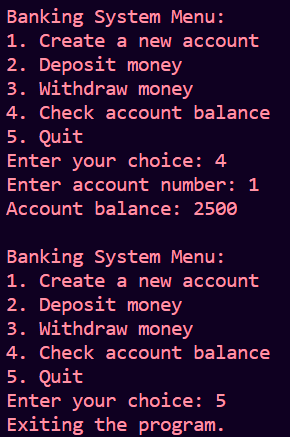

4. EXCEPTION HANDLING Design and implement a C++ program for a basic banking system with the following features:   
1. Create a BankAccount class that represents a bank account. Each account should have the following attributes: o Account holder's name o Account number (a unique identifier) o Current balance   
2. Implement exception classes for the following scenarios: o NegativeBalanceException: This exception should be thrown when a user attempts to withdraw more money than their account balance, resulting in a negative balance. o InvalidAmountException: This exception should be thrown when a user attempts to deposit or withdraw a negative amount or zero. 3. Your program should provide a menu-driven interface with the following options: o Create a new account. o Deposit money into an account. o Withdraw money from an account. o Check the account balance. o Quit the program.

4. Implement exception handling for the following situations: o Catch and handle exceptions related to negative balances and invalid amounts, ensuring that the program does not crash. o Display appropriate error messages when exceptions are caught. o Ensure that account balances are not modified when exceptions occur.

5. Ensure that the account numbers are unique for each account created.

6. Use appropriate object-oriented programming principles to design your classes and functions.

OUTPUT:  
  




TEMPLATE Design and implement a C++ program that provides generic data structures for managing students and courses in a university system. Your program should include the following components:   
1. Generic Linked List Template: Create a generic linked list template class called LinkedList that can store data of any data type. It should support basic operations such as insertion, deletion, and traversal. Demonstrate how to use this template class to create linked lists to store student and course data.   
2. Student Class: Create a Student class with the following attributes: o Student ID o Student name o GPA (Grade Point Average) Use the LinkedList template class to store a list of students.   
3. Course Class: Create a Course class with the following attributes: o Course code o Course name o Instructor name Use the LinkedList template class to store a list of courses.   
4. Menu-Driven Interface: Implement a user-friendly menu-driven interface that allows users to perform the following actions: o Add a student to the student list. o Add a course to the course list. o Display a list of students. o Display a list of courses. o Quit the program.   
5. Templates Usage: o Utilize templates for the LinkedList class to ensure it can work with both the Student and Course classes. o Demonstrate the creation of instances of LinkedList for both student and course data types.   
6. Error Handling: Implement error handling in your program using exception handling. For example, handle cases where the user tries to add a student to the student list but provides invalid input.

SOURCE CODE:

#include<bits/stdc++.h>

template <typename T>

class Node {

public:

    T data;

    Node\* next;

    Node(const T& data) : data(data), next(nullptr) {}

};

template <typename T>

class LinkedList {

private:

    Node<T>\* head;

    int size;

public:

    LinkedList() : head(nullptr), size(0) {}

    void insert(const T& data) {

        Node<T>\* newNode = new Node<T>(data);

        newNode->next = head;

        head = newNode;

        size++;

    }

    void remove(const T& data) {

        Node<T>\* current = head;

        Node<T>\* prev = nullptr;

        while (current != nullptr) {

            if (current->data == data) {

                if (prev == nullptr) {

                    head = current->next;

                } else {

                    prev->next = current->next;

                }

                delete current;

                size--;

                return;

            }

            prev = current;

            current = current->next;

        }

    }

    void display() {

        Node<T>\* current = head;

        while (current != nullptr) {

            std::cout << current->data << std::endl;

            current = current->next;

        }

    }

    int getSize() const {

        return size;

    }

    ~LinkedList() {

        while (head != nullptr) {

            Node<T>\* temp = head;

            head = head->next;

            delete temp;

        }

    }

};

class Student {

public:

    int studentID;

    std::string studentName;

    double gpa;

    Student(int id, const std::string& name, double gpa)

        : studentID(id), studentName(name), gpa(gpa) {}

    friend std::ostream& operator<<(std::ostream& os, const Student& student) {

        os << "Student ID: " << student.studentID << ", Name: " << student.studentName << ", GPA: " << student.gpa;

        return os;

    }

};

class Course {

public:

    std::string courseCode;

    std::string courseName;

    std::string instructorName;

    Course(const std::string& code, const std::string& name, const std::string& instructor)

        : courseCode(code), courseName(name), instructorName(instructor) {}

    friend std::ostream& operator<<(std::ostream& os, const Course& course) {

        os << "Course Code: " << course.courseCode << ", Name: " << course.courseName

           << ", Instructor: " << course.instructorName;

        return os;

    }

};

int main() {

    LinkedList<Student> studentList;

    LinkedList<Course> courseList;

    int choice;

    while (true) {

        std::cout << "\nUniversity System Menu:\n";

        std::cout << "1. Add a student\n";

        std::cout << "2. Add a course\n";

        std::cout << "3. Display students\n";

        std::cout << "4. Display courses\n";

        std::cout << "5. Quit\n";

        std::cout << "Enter your choice: ";

        std::cin >> choice;

        switch (choice) {

            case 1: {

                int id;

                std::string name;

                double gpa;

                try {

                    std::cout << "Enter student ID: ";

                    std::cin >> id;

                    if (std::cin.fail()) {

                        throw std::invalid\_argument("Invalid input for student ID.");

                    }

                    std::cout << "Enter student name: ";

                    std::cin.ignore();

                    std::getline(std::cin, name);

                    std::cout << "Enter GPA: ";

                    std::cin >> gpa;

                    if (std::cin.fail()) {

                        throw std::invalid\_argument("Invalid input for GPA.");

                    }

                    Student student(id, name, gpa);

                    studentList.insert(student);

                    std::cout << "Student added successfully." << std::endl;

                } catch (const std::invalid\_argument& e) {

                    std::cerr << "Error: " << e.what() << std::endl;

                    std::cin.clear();

                    std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

                }

                break;

            }

            case 2: {

                std::string code, name, instructor;

                std::cout << "Enter course code: ";

                std::cin >> code;

                std::cout << "Enter course name: ";

                std::cin.ignore();

                std::getline(std::cin, name);

                std::cout << "Enter instructor name: ";

                std::cin.ignore();

                std::getline(std::cin, instructor);

                Course course(code, name, instructor);

                courseList.insert(course);

                std::cout << "Course added successfully." << std::endl;

                break;

            }

            case 3:

                std::cout << "Students:\n";

                studentList.display();

                break;

            case 4:

                std::cout << "Courses:\n";

                courseList.display();

                break;

            case 5:

                std::cout << "Exiting the program." << std::endl;

                return 0;

            default:

                std::cerr << "Invalid choice. Please try again." << std::endl;

                break;

        }

    }

    return 0;

}

OUTPUT:  
