Name	ADITI RAO
UID no.	202220003
Experiment No.	9

AIM:	**Program on Exception Handling: Implement a Program to demonstrate Exception Handling**

Program 1

PROBLEM STATEMENT:

Consider the expression in the form a+b where ‘a’ and ‘b’ are numeric values and ‘+’ is any operator. Operators can be +,-,*,/,log and ^.

Write a program to handle these operations and the exceptions generated.

Exceptions that needs to be considered are :

a. Check if ‘a’ and ‘b’ are numbers. If yes then ok for further execution else throw an exception and handle it by asking the user to give correct inputs.
b. Check if ‘+’ is an operator as specified in the operators list.
c. Check for the order of the expression as ‘ a+b’ only and not +ab or ab+.
d. Check for divide by zero and log1 exceptions.

Write appropriate catch blocks for handling these exceptions. Make the program a menu driven one to ask the user for the operations to be performed.

ALGORITHM:

```
FUNCTION main()
PRINT "Operators can be  +, -, *, /, log and ^"
PRINT "Kindly enter the expression in the form of a + b, a - b, a * b, a / b, a log
b (for loga(b)) or a ^ b"
DECLARE char choice
DO
   CALL read()
   TRY
      IF checkNumeric() THEN
         IF checkOperatorValidity() THEN
            IF checkOperatorPosition() THEN
               CALL compute()
            ELSE
               THROW "Invalid expression. Incorrect operator position."
            END IF
```

```
                ELSE
                    THROW "Invalid expression. Unsupported operator."
                END IF
            ELSE
                THROW "Invalid expression. Non-numeric input."
            END IF
        CATCH const char* errorMsg
            PRINT "Error: " + errorMsg
        END TRY
        PRINT "Do you want to continue? (y/n): "
        READ choice
        IGNORE_LINE()
    WHILE choice == 'y' OR choice == 'Y'
    RETURN 0

FUNCTION read()
    PRINT "Kindly enter the expression: "
    READ str

FUNCTION checkZeroError()
    DECLARE pos = FIND '/' in str
    IF str[pos + 1] = '0' THEN
        THROW "Divide by zero error."

FUNCTION checkLogError()
    DECLARE pos = FIND "log" in str
    IF str[pos + 3] = '1' THEN
        THROW "Logarithm of 1 error."

FUNCTION checkNumeric()
    DECLARE hasNumeric = false
    FOR i = 0 to str.length() - 1
        IF IS_DIGIT(str[i]) THEN
            SET hasNumeric to true
        ELSE
            IF str[i] != '+' AND str[i] != '-' AND str[i] != '*' AND str[i] != '/' AND
str.substr(i, 3) != "log" AND str[i] != '^' THEN
                THROW "Invalid expression. Unsupported input."
            END IF
            IF str.substr(i, 3) = "log" THEN
                SET i to i + 2
            END IF
        END IF
```

```
    END FOR
    RETURN hasNumeric

FUNCTION checkOperatorValidity()
    IF str.find('+') != string::npos OR str.find('-') != string::npos OR str.find('/') !=
string::npos OR str.find('*') != string::npos OR str.find("log") != string::npos OR
str.find('^') != string::npos THEN
        IF str.find('/') != string::npos THEN
            CALL checkZeroError()
        ELSE IF str.find("log") != string::npos THEN
            CALL checkLogError()
        END IF
        RETURN true
    ELSE
        RETURN false

FUNCTION checkOperatorPosition()
    IF str[0] == '+' OR str[0] == '-' OR str[0] == '*' OR str[0] == '/' OR str[0] == 'l'
OR str[0] == '^' OR str[str.length() - 1] == '+' OR str[str.length() - 1] == '-' OR
str[str.length() - 1] == '*' OR str[str.length() - 1] == '/' OR str[str.length() - 1] == 'l'
OR str[str.length() - 1] == '^' THEN
        RETURN false
    ELSE
        RETURN true

FUNCTION compute()
    DECLARE opPos = -1
    FOR i = 0 to str.length() - 1
        IF str[i] == '+' OR str[i] == '-' OR str[i] == '*' OR str[i] == '/' OR str.substr(i,
3) == "log" OR str[i] == '^' THEN
            SET opPos to i
            BREAK
        END IF
    END FOR

    IF opPos == -1 THEN
        THROW "Invalid operator."
    END IF

    DECLARE a, b
    TRY
        a = stoi(str.substr(0, opPos))
        b = stoi(str.substr(opPos + 1))
```

	CATCH invalid_argument THROW "Invalid operands." DECLARE op = str[opPos] SWITCH op CASE '+' PRINT a + b CASE '-' PRINT a - b CASE '*' PRINT a * b CASE '/' PRINT a / b CASE 'l' PRINT log(b) / log(a) CASE '^' PRINT pow(a, b) END SWITCH

PROGRAM:	#include <iostream>
	#include <string>
	#include <cctype>
	#include <cmath>
	
	using namespace std;
	
	static string str;
	
	void read()
	{
	cout << "Kindly enter the expression: ";
	getline(cin, str);
	}
	
	void checkZeroError()
	{
	int pos = str.find('/');
	if (str[pos + 1] == '0') {
	throw "Divide by zero error.";
	}
	}

```
void checkLogError()
{
    int pos = str.find("log");
    if (str[pos + 3] == '1' && str[pos + 4] == '\0') {
        throw "Logarithm of 1 error.";
    }
}

bool checkNumeric()
{
    bool hasNumeric = false;
    for (int i = 0; i < str.length(); i++) {
        if (isdigit(str[i])) {
            hasNumeric = true;
        }
        else {
            if (str[i] != '+' && str[i] != '-' && str[i] != '*' && str[i] != '/' && str.substr(i, 3) !=
"log" && str[i] != '^') {
                if (isalpha(str[i])) {
                    throw "Invalid expression. Non-numeric input.";
                }
                else {
                    throw "Invalid expression. Unsupported operator.";
                }
            }
            if (str.substr(i, 3) == "log") {
                i += 2;
            }
        }
    }
    return hasNumeric;
}

bool checkOperatorValidity()
{
    if ((str.find('+') != string::npos) || (str.find('-') != string::npos) || (str.find('/') !=
string::npos) || (str.find('*') != string::npos) || (str.find("log") != string::npos) || (str.find('^')
!= string::npos)) {
        if (str.find('/') != string::npos) {
            checkZeroError();
        }
        else if (str.find("log") != string::npos) {
            checkLogError();
```

```
        }
        return true;
    }
    else {
        return false;
    }
}

bool checkOperatorPosition()
{
    if (str[0] == '+' || str[0] == '-' || str[0] == '*' || str[0] == '/' || str[0] == 'l' || str[0] == '^' ||
str[str.length() - 1] == '+' || str[str.length() - 1] == '-' || str[str.length() - 1] == '*' ||
str[str.length() - 1] == '/' || str[str.length() - 1] == 'l' || str[str.length() - 1] == '^') {
        return false;
    }
    else {
        return true;
    }
}

void compute()
{
    int opPos = -1;
    for (int i = 0; i < str.length(); i++) {
        if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/' || str.substr(i, 3) == "log" || str[i]
== '^') {
            opPos = i;
            break;
        }
    }

    if (opPos == -1) {
        throw "Invalid operator.";
    }

    int a, b;
    try {
        if (str.substr(opPos, 3) == "log") {
            a = stoi(str.substr(0, opPos));
            b = stoi(str.substr(opPos + 3));
        }
        else {
            a = stoi(str.substr(0, opPos));
```

```
            b = stoi(str.substr(opPos + 1));
        }
    }
    catch (const invalid_argument) {    // Catches invalid_argument exception
        throw "Invalid operands.";
    }

    char op = str[opPos];
    switch (op) {
        case '+':
            cout << (a + b) << endl;
            break;
        case '-':
            cout << (a - b) << endl;
            break;
        case '*':
            cout << (a * b) << endl;
            break;
        case '/':
            cout << ((float)a / b) << endl;
            break;
        case 'l':
            cout << ((float)log(b) / log(a)) << endl;
            break;
        case '^':
            cout << pow(a, b) << endl;
            break;
    }
}



int main()
{
    cout << "Operators can be  +, -, *, /, log and ^" << endl;
    cout << "Kindly enter the expression in the form of a + b, a - b, a * b, a / b, a log b (for
loga(b)) or a ^ b" << endl;
    char choice;
    do {
        read();
        try {
            if (checkNumeric()) {
                if (checkOperatorValidity()) {
```

```
                if (checkOperatorPosition()) {
                    compute();
                }
                else {
                    throw "Invalid expression. Incorrect operator position.";
                }
            }
            else {
                throw "Invalid expression. Unsupported operator.";
            }
        }
        else {
            throw "Invalid expression. Non-numeric input.";
        }
    }
    catch (const char* errorMsg) {
        cout << "Error: " << errorMsg << endl;
    }
    cout << "Do you want to continue? (y/n): ";
    cin >> choice;
    cin.ignore();
  }
  while (choice == 'y' || choice == 'Y');

  return 0;
}
```

RESULT:

```
Kindly enter the expression in the form of a + b, a - b, a * b, a / b, a log b (for loga(b)) or a ^ b
Kindly enter the expression: 2+3
5
Do you want to continue? (y/n): y
Kindly enter the expression: 2-3
-1
Do you want to continue? (y/n): y
Kindly enter the expression: 12*13
156
Do you want to continue? (y/n): y
Kindly enter the expression: 12/25
0.48
Do you want to continue? (y/n): y
```

```
Kindly enter the expression: 12log144
2
Do you want to continue? (y/n): y
Kindly enter the expression: 2^4
16
Do you want to continue? (y/n): y
Kindly enter the expression: +24
Error: Invalid expression. Incorrect operator position.
Do you want to continue? (y/n): y
Kindly enter the expression: a+7
Error: Invalid expression. Non-numeric input.
Do you want to continue? (y/n): y
Kindly enter the expression: 1$4
Error: Invalid expression. Unsupported operator.
```

```
Kindly enter the expression: 1/0
Error: Divide by zero error.
Do you want to continue? (y/n): y
```

```
Kindly enter the expression: 2log14
3.80736
Do you want to continue? (y/n): y
Kindly enter the expression: 2log1
Error: Logarithm of 1 error.
```

Program 2

PROBLEM STATEMENT:

Write a program that converts dates from numerical month/day format to alphabetic month/day (for example 1/31 or 01/31 corresponds to January 31).
You will define two exception classes, one called MonthError and another called DayError. If the user enters anything other than a legal month number (integers from 1 to 12), then your program will throw and catch a MonthError. Similarly, if the user enters anything other than a valid day number (integers from 1 to either 29, 30, or 31, depending on the month), then your program will throw and catch a DayError. To keep things simple, always allow 29 days for February. (If the user enters an illegal month or day other than the valid number, for example, some gibberish like 8&*68, the program must still print a MonthError/DayError as applicable)

Sample output:
Enter Date in month/day numeric notation:
1/30
That is the same as
January 30
Again? (y/n)
y

Enter Date in month/day numeric notation:
02/29
That is the same as

	February 29 Again? (y/n) y Enter Date in month/day numeric notation: 02/30 Invalid day for the corresponding month Try Again! Enter Date in month/day numeric notation: 1@12/23 Invalid month Try Again! Again? (y/n) y Enter Date in month/day numeric notation: 1@12&23 Invalid Date Try Again! Again? (y/n) n End of program.
ALGORITHM:	class MonthError extends Exception method what() return "Invalid month" class DayError extends Exception method what() return "Invalid day for the corresponding month" function getMonthName(month) static monthNames = {"", "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"} if month < 1 or month > 12 throw MonthError return monthNames[month] function isValidDay(month, day)

```
    static daysInMonth = {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

    if day < 1 or day > daysInMonth[month]
        throw DayError

    return true

function isSpecialCharacter(ch)
    static specialCharacters = "!@#$%^&*()_+=-{}[]|\\:;<>,.?~`"

    if ch is found in specialCharacters
        return true

    return false

function main()
    choice = 'y'

    while choice is 'y' or choice is 'Y'
        month, day = 0, 0
        slash = '\0'

        display "Enter Date in month/day numeric notation: "
        read month, slash, day

        try
            if isSpecialCharacter(slash)
                throw runtime_error("Invalid date format")

            monthName = getMonthName(month)
            isValidDay(month, day)

            display "That is the same as"
            display monthName, day
        catch MonthError as ex
            display ex.what()
            display "Try Again!"
            continue
        catch DayError as ex
            display ex.what()
            display "Try Again!"
        catch exception as ex
            display "Invalid input. Special character other than '/' is not allowed."
```

	display "Try Again!" ignore the rest of the input display "Again? (y/n): " read choice display "End of program."

PROGRAM:	#include <iostream>
	#include <string>
	#include <exception>
	#include <limits>
	using namespace std;
	
	/*As exception is already a pre-defined class in C++, we can simply use it by inheriting it from the exception class. This is called user-defined exception. We can then use functionalities limited to
	exception class by this inheritance*/
	
	class MonthError : public exception {
	public:
	  const char* what() const throw() {
	  return "Invalid month";
	  }
	};
	
	class DayError : public exception {
	public:
	  const char* what() const throw() {
	  return "Invalid day for the corresponding month";
	  }
	};
	
	// Function to convert numerical month to alphabetic month
	string getMonthName(int month) {
	  static const string monthNames[] = {"", "January", "February", "March", "April","May", "June", "July", "August","September", "October", "November", "December"};
	
	  if (month < 1 || month > 12)
	  throw MonthError();
	
	  return monthNames[month];

```
}

// Function to check if the day is valid for the corresponding month
bool isValidDay(int month, int day) {
    static const int daysInMonth[] = {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    if (day < 1 || day > daysInMonth[month])
        throw DayError();

    return true;
}

// Function to check if the character is a special character other than slash
bool isSpecialCharacter(char ch) {
    static const string specialCharacters = "!@#$%^&*()_+=-{}[]|\\:;<>,.?~`";
    return specialCharacters.find(ch) != string::npos;
}

int main()
{
    char choice;
    do {
        int month, day;
        char slash;

        cout << "Enter Date in month/day numeric notation: ";
        cin >> month >> slash >> day;                    //slash is used to store the '/' character

        try {
            if (isSpecialCharacter(slash))
                throw runtime_error("Invalid date format"); //we can use runtime_error because
we inherited MonthError and DayError from exception class
            string monthName = getMonthName(month);
            isValidDay(month, day);
            cout << "That is the same as" << endl;
            cout << monthName << " " << day << endl;
        }
        catch (const MonthError ex) {
            cout << ex.what() << endl;
            cout << "Try Again!" << endl;
            continue;
        }
        catch (const DayError ex) {
```

```
            cout << ex.what() << endl;
            cout << "Try Again!" << endl;
        }
        catch (const exception ex) {
            cout << "Invalid input. Special character other than '/' is not allowed." << endl;
            cout << "Try Again!" << endl;
            cin.ignore(numeric_limits<streamsize>::max(),'\n'); //to ignore the rest of the input
        }
        cout << "Again? (y/n): ";
        cin >> choice;
    }
    while (choice == 'y' || choice == 'Y');

    cout << "End of program." << endl;

    return 0;
}
```

RESULT:

```
Enter Date in month/day numeric notation: 1/30
That is the same as
January 30
Again? (y/n): y
Enter Date in month/day numeric notation: 02/29
That is the same as
February 29
Again? (y/n): y
Enter Date in month/day numeric notation: 02/30
Invalid day for the corresponding month
Try Again!
Again? (y/n): y
```

```
Enter Date in month/day numeric notation: 1@12/30
Invalid input. Special character other than '/' is not allowed.
Try Again!
Again? (y/n): y
Enter Date in month/day numeric notation: 1@12#30
Invalid input. Special character other than '/' is not allowed.
Try Again!
Again? (y/n): y
Enter Date in month/day numeric notation: 9/20
That is the same as
September 20
Again? (y/n): n
End of program.
```

CONCLUSION	Exception handling in C++ is a powerful mechanism that allows developers to gracefully handle and recover from runtime errors. By using try-catch blocks, custom exception classes, and the inheritance hierarchy, programmers can effectively manage and communicate errors during program execution. Exception handling enhances code robustness, promotes maintainability, and ensures a more reliable and predictable program flow, leading to more robust and resilient C++ applications.