TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

Universidad Autónoma de Guadalajara (U.A.G)

Estudiante:

Fernando Orozco Ortiz de la Peña

Semestre & Parcial:

Tercer Semestre/Primer Parcial

Materia:

Programación Orientada a Objetos (POO)

Maestro:

Manuel Balderas Victoria

Maestria:

Ingeniería en Software y Minería de Datos

Fecha:

7 de Noviembre del 2025 -> 9 de Noviembre del 2025

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

Para este tercer reporte de Programación Orientada a Objetos elaborado por el estudiante Fernando Orozco Ortiz de la Peña, se explicara y elaborará los ejercicios de laboratorio que se encontraron en el segundo módulo de Cisco Networking Academy. Se explicará cada objetivo, resultado, y característica de lo encontrado y elaborado de cada uno de los programas que los ejercicios de Laboratorio nos dieron como ejemplos.

EJEMPLO 1: "Exceptions: input validation"

Para el primer ejercicio, se trata de familiarizarnos con el uso de excepciones en C++, osea tendremos que simplemente identificar las situaciones en las que se generan excepciones durante la ejecución de un programa, a lado de aplicar correctamente las técnicas de manejo de excepciones mediante el uso de bloques try, catch y la instrucción throw.

```cpp
#include <iostream>
#include <stdexcept>

using namespace std;

int main() {
    try {
        int x;
        cout << "Enter a number: ";
        cin >> x;

        if (cin.fail()) {
            throw invalid_argument("Invalid input. Not a number.");
        }

        if (x == 0) {
            throw invalid_argument("Your input is not valid, you can't divide by zero.");
        }

        cout << x / 2 << endl;
    }
    catch (const invalid_argument& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

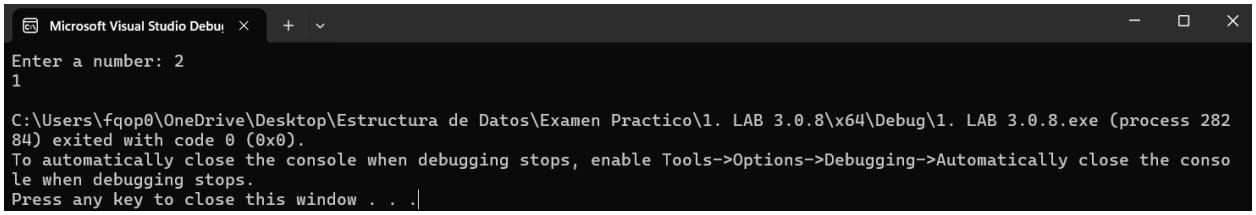
Primero dentro de la Int Main se tiene que insertar un método try para que las excepciones se puedan generar, luego dentro del método try se tendrá que poner un int x para que se pueda insertar un valor que se intenta almacenar en x, pero también se agregara una función que se encargará que indicar si el usuario introduce algo que no puede convertirse a entero, la función cin.fail() devolverá true como resultado.

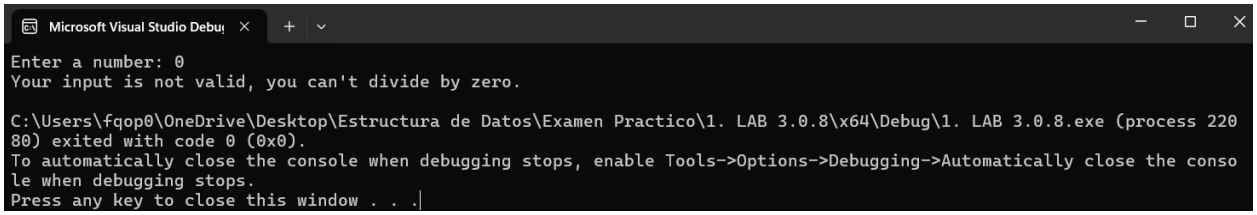
TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

No solamente eso, pero si el usuario introduce 0, se lanza manualmente una excepción para evitar una división inválida.

Finalmente se usará el método catch para que el código pueda manejar las excepciones a lado de hacer que el programa no se detenga y que imprima el mensaje de error cuando se ejecuta el bloque catch.

Asi quedaria el resultado final de este ejercicio de cisco:





EJEMPLO 2: "Exceptions: divide by zero"

Para este ejercicio, este se encargará de ayudarnos a manejar las excepciones en C++, a lado de enseñar cómo detectar situaciones en las que se lanzan excepciones y cómo capturarlas adecuadamente para evitar errores en tiempo de ejecución, especialmente en operaciones aritméticas

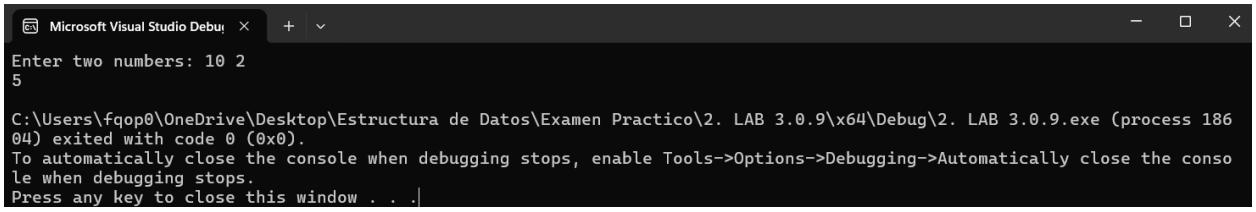
```cpp
#include <iostream>
#include <stdexcept>

int main() {
    double a, b;

    std::cout << "Enter two numbers: ";
    std::cin >> a >> b;

    try {
        if (b == 0) {
            throw std::runtime_error("Your input is not valid, you can't divide by zero.");
        }

        double result = a / b;
        std::cout << result << std::endl;
    }
    catch (const std::runtime_error& e) {
        std::cout << e.what() << std::endl;
    }

    return 0;
}
```

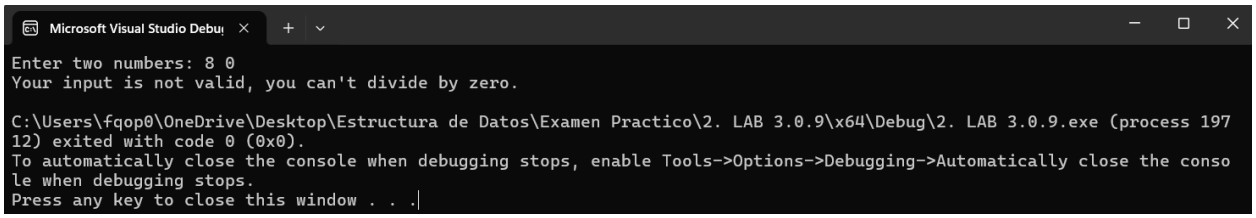
TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

Primero se tendrá que hacer algo similar con el ejercicio anterior, usando los métodos try y catch respectivamente para la generación y manejamiento de excepciones en el código/programa.

Dentro del método try,también se tendrá que usar el método throw, el cual se encargará de lanzar una excepción específica si ocurre una condición no válida, este se utiliza para crear y lanzar una excepción del tipo runtime_error, el cual representa un error durante la ejecución.

Todo eso hace que el resultado sea lo siguiente dentro del programa para este ejercicio de cisco:





EJEMPLO 3: “Exceptions: input validation”

Para este ejercicio, se trata de familiarizarse con identificando situaciones en las que deben lanzarse excepciones, agregar códigos para no solamente manejar las excepciones, pero también para lanzarlos.

```cpp
#include <iostream>
#include <stdexcept>
using namespace std;

double div(double a, double b) {
    try {
        if (b == 0)
            throw runtime_error("Are you kidding me? Your input is not valid. You can't divide by zero.");
        return a / b;
    }
    catch (const runtime_error& e) {
        cout << e.what() << endl;
        return 0; // Valor arbitrario, ya que ocurrió un error.
    }
}

int main() {
    double x, y;
    cin >> x >> y;
    cout << div(x, y) << endl;
    return 0;
}

```

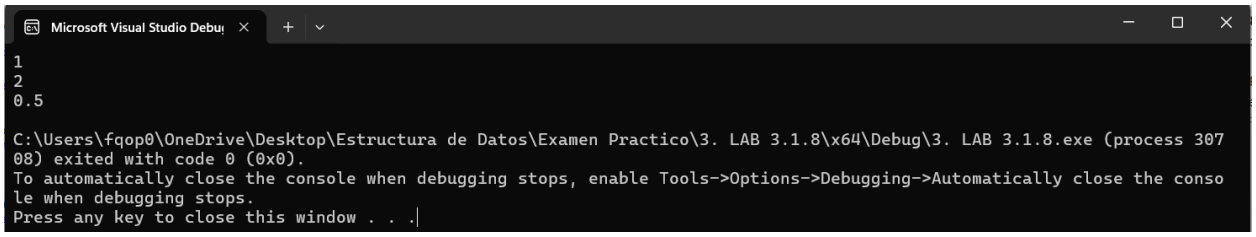
TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

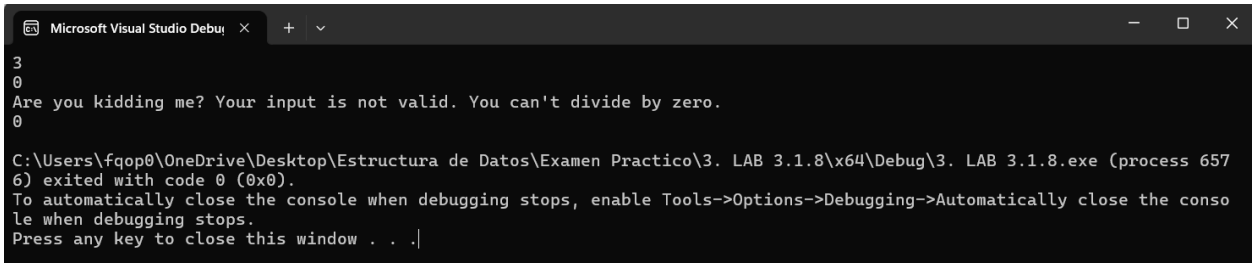
Primero se tendrá que utilizar el método double para que se pueda almacenar valores con decimales. Luego se tendrá que insertar los bloques try y catch, donde:

- El bloque try del programa verifica si el denominador b es igual a cero, donde se lanza una excepción de tipo runtime_error con un mensaje personalizado.
- El bloque catch captura esa excepción y muestra el mensaje de error, en el caso de que si muestre un error la función devolverá el valor como 0.

Finalmente en la Int Main, se solicitará al usuario que ingrese dos valores reales (dependiendo si es 0 o no), donde llamara la función div(x, y) para poder después imprimir el resultado de la pantalla.

Con todo eso en mente, este será el resultado final de este ejercicio de cisco:





EJEMPLO 4: “Exceptions: catch block”

Para este ejemplo, se trata de la creación de clases de excepciones personalizadas en la forma de enviar información adicional al lanzar una excepción.

En este caso es simplemente buscar el área de distintas figuras como cuadrados y rectángulos para poder aprender más sobre el manejamiento de excepciones.

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

```cpp
#include <iostream>
#include <string>
using namespace std;

class InvalidDimensionException {
private:
    string message;
public:
    InvalidDimensionException(const string& msg) : message(msg) {}

    string getMessage() const {
        return message;
    }
};

double areaCuadrado(double lado) {
    if (lado <= 0) {
        throw InvalidDimensionException("Your input is not valid. The area can't be negative.");
    }
    return lado * lado;
}

double areaRectangulo(double base, double altura) {
    if (base <= 0 || altura <= 0) {
        throw InvalidDimensionException("Your input is not valid. The area can't be negative.");
    }
    return base * altura;
}
```

Dentro de la clase “InvalidDimensionException” se trata como dice el nombre de la clase, definir una excepción específica para el caso de dimensiones inválidas. Donde lo que queremos realizar es encapsular un mensaje de error para nuestro programa donde el atributo message se encarga de guardar la descripción del error en si, mientras tanto el metodo getMessage se encarga de retornar dicho mensaje para que funcione.

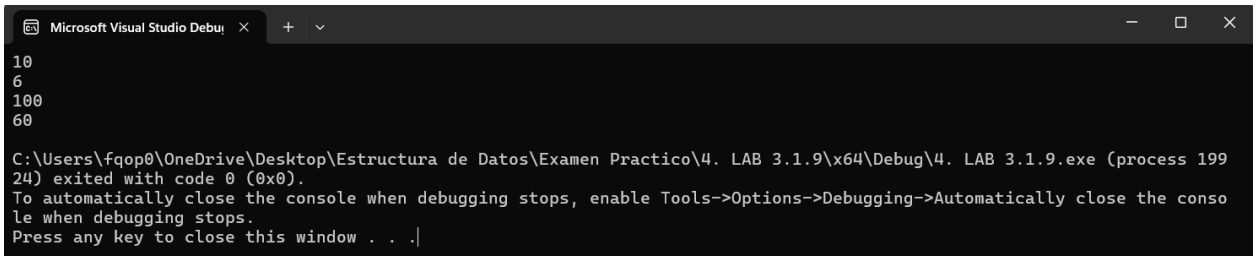
Luego con los datos double areaCuadrado y double areaRectangulo respectivamente, estos dos se encargaran de identificar si el lanzamiento de un encapsulamiento si el valor que contiene es menor o igual que 0 o si las dimensiones de la figura son inválidas.

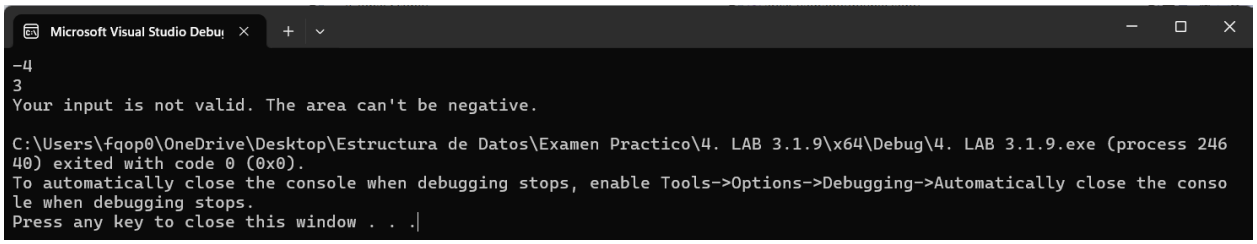
```cpp
int main() {
    double a, b;

    cin >> a;
    cin >> b;

    try {
        double area1 = areaCuadrado(a);
        double area2 = areaRectangulo(a, b);

        cout << area1 << endl;
        cout << area2 << endl;
    }
    catch (InvalidDimensionException& e) {
        cout << e.getMessage() << endl;
    }

    return 0;
```

Finalmente con la Int Main, este se encargará de solicitar dos distintos valores para representar Lado*Lado y Base*Altura respectivamente para obtener el resultado final.

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

Con todo eso en mente, este sería el resultado final de este ejemplo de cisco:





EJEMPLO 5: "Exceptions: simple checks" (Partes 1 y 2)

PARTE 1

Para esta primera parte del quinto ejemplo, este se trata lo mismo que se hizo con las actividades anteriores, sobre el lanzamiento de excepciones y su manejos.

Pero en este caso, nuestro objetivo es crear un objeto matriz con dimensiones específicas donde tengamos que insertar valores que representen

```cpp
class Matrix {
private:
    int rows, cols;
    vector<vector<int>> data;

public:
    Matrix() : rows(0), cols(0) {}

    Matrix(int r, int c) : rows(r), cols(c) {
        if (r <= 0 || c <= 0)
            throw runtime_error("Matrix dimensions must be positive");
        data.assign(rows, vector<int>(cols, 0));
    }

    void fillMatrix() {
        if (rows == 0 || cols == 0)
            throw runtime_error("Matrix is empty");

        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                cout << "Ingrese valor [" << i << "][" << j << "]: ";
                cin >> data[i][j];
            }
        }
    }

    void print() const {
        if (rows == 0 || cols == 0) {
            cout << "Empty matrix" << endl;
            return;
        }
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j)
                cout << data[i][j] << " ";
            cout << endl;
        }
    }

    void addValue(int val) {
        if (rows == 0 || cols == 0)
            throw runtime_error("Matrix is empty");

        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < cols; ++j)
                data[i][j] += val;
    }

    Matrix addMatrix(const Matrix& other) const {
        if (rows == 0 || cols == 0 || other.rows == 0 || other.cols == 0)
            throw runtime_error("Matrix is empty");

        if (rows != other.rows || cols != other.cols)
            throw runtime_error("Matrices don't fit.");

        Matrix result(rows, cols);
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < cols; ++j)
                result.data[i][j] = data[i][j] + other.data[i][j];

        return result;
    }
};
```

La clase Matrix representa una matriz bidimensional de enteros y encapsula sus operaciones básicas, donde:

- En los atributos privados la "int row, cols" representa el número de filas y columnas que contiene la matriz en sí.
- En los atributos públicos, se utilizarán constructores que se encargaran de no solamente crear una matriz vacía o de tamaño respectivamente, sino también se encargará de lanzar una excepción si las dimensiones resultantes son inválidas.

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

Y hablando de del atributo público, también incluirá las palabras clave “Void” y “Matrix” que se encargaran de:

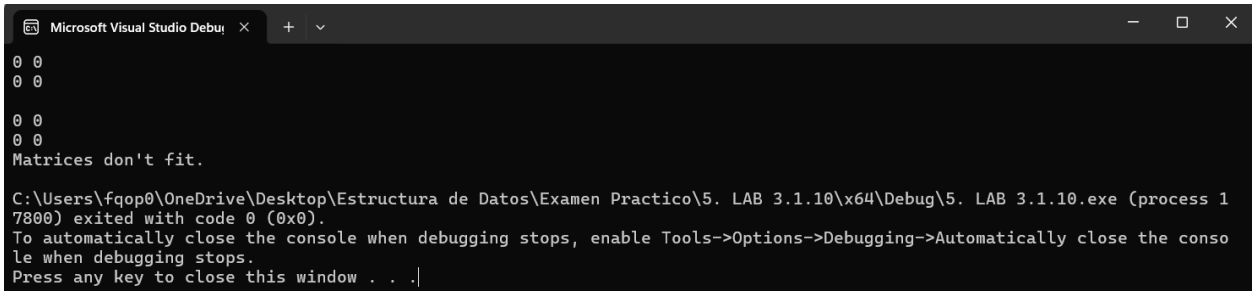
- En el caso de la Void; Permite al usuario llenar la matriz desde el teclado, muestra el contenido de la matriz en consola, y suma un mismo valor (val) a todos los elementos de la matriz.
- Mientras tanto en el caso de Matrix; Ayuda a realizar la suma de elemento a elemento de dos matrices para poder lanzar una excepción si alguna de las matrices está vacía o si las dimensiones no coinciden con sigo mismo.

```
69  int main() {
70      try {
71          Matrix m1(2, 2);
72          Matrix m2(2, 2);
73          Matrix m3(3, 3);
74
75          m1.print();
76          cout << endl;
77
78          try {
79              Matrix result1 = m1.addMatrix(m2);
80              result1.print();
81          }
82          catch (const exception& e) {
83              cout << e.what() << endl;
84          }
85
86          try {
87              Matrix result2 = m1.addMatrix(m3);
88              result2.print();
89          }
90          catch (const exception& e) {
91              cout << e.what() << endl;
92          }
93
94      }
95      catch (const exception& e) {
96          cerr << "Error: " << e.what() << endl;
97      }
98
99      return 0;
100 }
```

Finalmente en la Int Main, aquí se tendrá que crear tres distintas matrices (con los primeros dos matrices siendo 2x2 mientras el último es de 3x3).

Ya que las primeras dos matrices son de la misma cantidad, esas dos coinciden perfectamente, pero ya que la última matriz es de 3x3 en vez de 2x2, se lanzará la excepción de error.

Con todo eso en mente, esta sería el primer resultado de esta actividad del ejemplo 5:



TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

PARTE 2

Para la segunda parte de este ejemplo, tendremos que implementar una clase simple que se encargará de almacenar un número entero junto con sus límites mínimos y máximos que se le puede permitir, donde la clase ayuda a controlar que el valor no sobrepase esos mismos límites cuando se realicen una operación de suma o resta.

```
5   class LimitedValue {
6   private:
7       int value;
8       int minLimit;
9       int maxLimit;
10
11  public:
12      LimitedValue(int val, int minL, int maxL) {
13          if (minL > maxL)
14              throw invalid_argument("Invalid limits: min > max");
15          if (val < minL || val > maxL)
16              throw invalid_argument("Initial value out of range");
17          value = val;
18          minLimit = minL;
19          maxLimit = maxL;
20      }
21
22      void add(int x) {
23          if (value + x > maxLimit)
24              throw out_of_range("Value could exceed limit.");
25          value += x;
26      }
27
28      void subtract(int x) {
29          if (value - x < minLimit)
30              throw out_of_range("Value could exceed limit.");
31          value -= x;
32      }
33
34      void display() const {
35          cout << value << endl;
36      }
37  };
```

Primero se creará la clase LimitedValue, dónde se encargará de almacenar el valor actual del objeto, y ayudará a definir los límites inferior y superior permitidos dentro del atributo privado.

Mientras tanto en el atributo público, este se encargara de crear un constructor que ayudará a inicializar el objeto con un valor inicial y los límites que puede tener, a lado de verificar dos distintas condiciones donde el límite mínimo no sea mayor que el máximo y que el valor inicial este dentro de los límites, si no las cumple entonces lanzará una excepción de tipo “invalid_argument”.

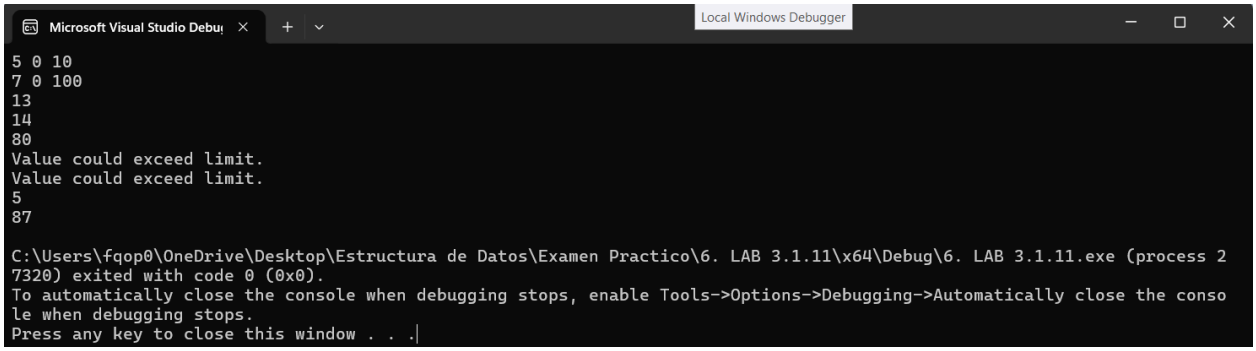
Luego con la palabra clave “Void”, este ayudará a representar el resultado que se mostrará en pantalla dependiendo si el resultado sea de una suma o de una resta.

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

```
39  int main() {
40      try {
41          int v1, min1, max1;
42          int v2, min2, max2;
43          int add1, add2, add3;
44
45          cin >> v1 >> min1 >> max1;
46          cin >> v2 >> min2 >> max2;
47          cin >> add1 >> add2 >> add3;
48
49          LimitedValue obj1(v1, min1, max1);
50          LimitedValue obj2(v2, min2, max2);
51
```

Finalmente con la Int Main, este se encargará de que el programa reciba dos conjuntos de valores iniciales y límites, y luego tres valores adicionales para sumar.

Con todo esto en mente, este sería el segundo ejemplo de quinto ejercicio de cisco:



EJEMPLO 6: “Exceptions: file checks”

El propósito de este ejercicio es desarrollar una clase que representa una matriz 2x2 utilizando métodos para poder cargar y guardar la matriz en un archivo, donde los archivos se puedan simular y manejar errores comunes.

```
8   class Matrix2x2 {
9   private:
10      double data[2][2];
11
12  public:
13      Matrix2x2(double a11 = 0, double a12 = 0, double a21 = 0, double a22 = 0) {
14          data[0][0] = a11; data[0][1] = a12;
15          data[1][0] = a21; data[1][1] = a22;
16      }
17
18      void display() const {
19          cout << "[" << data[0][0] << ", " << data[0][1] << "]" << endl;
20          cout << "[" << data[1][0] << ", " << data[1][1] << "]" << endl;
21      }
22
23      void loadFromFile(const string& path) {
24          ifstream file;
25          file.open(path);
26          if (!file.is_open()) {
27              cerr << "File not found at: " << path << endl;
28              throw runtime_error("File not found");
29          }
30
31          try {
32              for (int i = 0; i < 2; ++i)
33                  for (int j = 0; j < 2; ++j)
34                      file >> data[i][j];
35
36              if (file.fail())
37                  throw runtime_error("File format error");
38
39              file.close();
40          }
41          catch (...) {
42              file.close();
43              throw; // Re-lanza la excepción
44          }
45      }
```

Primero se creara la clase Matrix2x2 donde tendra dos atributos, uno publico y uno privado.

En el atributo privado se utilizara el double data[2][2] para poder guardar los valores numéricos de la matriz en un arreglo bidimensional.

Luego en el atributo público se creará un constructor que se

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

```
47      void saveToFile(const string& path) const {
48          ofstream file;
49          file.open(path);
50          if (!file.is_open()) {
51              cerr << "No rights to write to file: " << path << endl;
52              throw runtime_error("No write access");
53          }
54
55          try {
56              file << data[0][0] << " " << data[0][1] << endl;
57              file << data[1][0] << " " << data[1][1] << endl;
58              file.close();
59          }
60          catch (...) {
61              file.close();
62              throw;
63          }
64      }
65  };
```

encargará de inicializar la matriz con los valores dados al sistema. Después con el atributo void, se encargará de hacer lo siguiente:

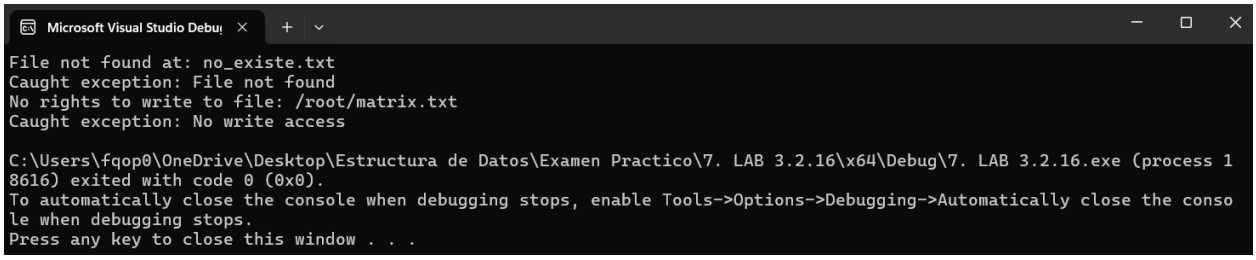
- Ayudará a mostrar la matriz en formato legible por consola.
- Ayudará a leer los valores de la matriz desde un archivo de texto.
- Ayudará a guardar los valores actuales de la matriz en un archivo de texto.

```
int main() {
    Matrix2x2 m(1, 2, 3, 4);

    try {
        m.loadFromFile("no_existe.txt");
    }
    catch (const exception& e) {
        cout << "Caught exception: " << e.what() << endl;
    }

    try {
        m.saveToFile("/root/matrix.txt");
    }
    catch (const exception& e) {
        cout << "Caught exception: " << e.what() << endl;
    }

    return 0;
}
```

Finalmente con la Int Main, este se encargará de demostrar el funcionamiento de la clase y el manejo de excepciones que presenta, donde se creará una matriz con valores iniciales y lanzará una excepción ya que el archivo en sí no existe.

Con todo eso en mente, este sería el resultado final de este ejercicio de cisco:

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS



EJEMPLO 7: "Exceptions: checking strings"

```cpp
#include <iostream>
#include <sstream>
#include <vector>
#include <stdexcept>

class IPAddress {
    int octets[4];

public:
    IPAddress(const std::string& ip) {
        std::stringstream ss(ip);
        std::string part;
        int i = 0;

        while (std::getline(ss, part, '.')) {
            if (i >= 4)
                throw std::invalid_argument("Exception - invalid IP number.");

            int num;
            try {
                num = std::stoi(part);
            }
            catch (...) {
                throw std::invalid_argument("Exception - invalid IP number.");
            }

            if (num < 0 || num > 255)
                throw std::invalid_argument("Exception - invalid IP number.");

            octets[i++] = num;
        }

        if (i != 4)
            throw std::invalid_argument("Exception - invalid IP number.");
    }
```

Primero se creará una clase llamada "IPAddress", la cual se encargará de crear un constructor que recibirá una dirección IP en formato string. No solamente eso pero también se encargará de hacer lo siguiente:

- Se encargará de usar la std::Stringstream y getline para poder dividir la IP en sus 4 octetos.
- Se encargará de convertir cada parte en un número entero con std::stoi.
- Y verificará que solamente haya 4 octetos y que cada uno de ellos sea del rango de 0 a 255.

Luego con el método print(int range, este ayudará a imprimir una secuencia de direcciones IP consecutivas.

No solamente eso pero hará que el rango sea mayor que 0, menor o igual que 256 y tendrá una potencia de 2 (eso siendo la condición (range & (range - 1)) != 0), a lado de generar las IPs Consecutivas que se encargaran de copiar los octetos originales

```cpp
    void print(int range) const {

        if (range <= 0 || range > 256 || (range & (range - 1)) != 0)
            throw std::invalid_argument("Exception - invalid range.");

        for (int i = 1; i < range; i++) {
            int newOctets[4];
            for (int j = 0; j < 4; ++j)
                newOctets[j] = octets[j];

            newOctets[3] += i;

            for (int j = 3; j > 0; --j) {
                if (newOctets[j] > 255) {
                    newOctets[j] -= 256;
                    newOctets[j - 1]++;
                }
            }

            if (newOctets[0] > 255)
                break;

            std::cout << newOctets[0] << "."
                << newOctets[1] << "."
                << newOctets[2] << "."
                << newOctets[3] << std::endl;
        }
    }
};
```

en un nuevo arreglo newOctets donde se sumará i al último octeto.

Finalmente con la Int Main, este se encargará de definir un vector con las siguientes pruebas:

```
67  int main() {
68      std::vector<std::pair<std::string, int>> tests = {
69          {"212.111.212.128", 2},
70          {"212.112.212.128", 4},
71          {"212.113.212.128", 6},
72          {"212.114.212.328", 4}
73      };
```

Donde no solamente cada elemento tiene una dirección IP y un rango, pero el ciclo for ayudará a crear un objeto IPAddress donde se capturará la excepción y mostrará un mensaje de error si la IP o el Rango son inválidos.

Con todo eso en mente, este sería el resultado final de este ejemplo de Cisco:

```
Microsoft Visual Studio Debu
212.111.212.129

212.112.212.129
212.112.212.130
212.112.212.131

Exception - invalid range.
Exception - invalid IP number.

C:\Users\fqop0\OneDrive\Desktop\Estructura de Datos\Examen Practico\8. LAB 3.2.17\x64\Debug\8. LAB 3.2.17.exe (process 1
8960) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

EJEMPLO 8: “Exceptions: including information in exceptions”

```
1   #include <iostream>
2   #include <sstream>
3   #include <vector>
4   #include <stdexcept>
5
6   class IPHeader {
7   private:
8       std::string sourceIP;
9       std::string destinationIP;
10
11      bool checkIP(const std::string& ip) {
12          std::stringstream ss(ip);
13          std::string segment;
14          int count = 0;
15
16          while (std::getline(ss, segment, '.')) {
17              if (segment.empty()) return false;
18              if (segment.size() > 3) return false;
19              for (char c : segment) {
20                  if (!isdigit(c)) return false;
21              }
22              int num = std::stoi(segment);
23              if (num < 0 || num > 255) return false;
24              ++count;
25          }
26
27          return (count == 4);
28      }
```

Primero se creará una clase llamada “IPHeader”, la cual tendrá dos atributos, uno privado y uno público.

En el atributo privado, este se encargará de guardar las direcciones IP de origen y destino, mientras sus métodos se encargan de validar si una dirección IP tiene el formato correcto.

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

El formato en si será correcto si tiene lo siguiente:

- Si está compuesta por 4 números separados por puntos (.)
- Si cada número debe estar en el rango de 0 a 255
- Y si no contiene caracteres no numéricos

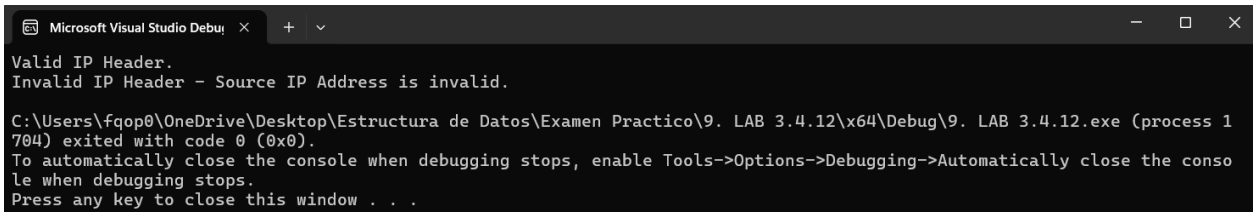
```
30  public:
31      IPHeader(const std::string& src, const std::string& dest)
32          : sourceIP(src), destinationIP(dest)
33      {
34          if (!checkIP(sourceIP))
35              throw std::invalid_argument("Invalid IP Header - Source IP Address is invalid.");
36          if (!checkIP(destinationIP))
37              throw std::invalid_argument("Invalid IP Header - Destination IP Address is invalid.");
38      }
39
40      void print() const {
41          std::cout << "Valid IP Header." << std::endl;
42      }
43  };
```

Luego en la sección pública, este se encargará de insertar un constructor que ayudará a recibir dos cadenas (src y dest) y las asigna a los atributos sourceIP y destinationIP, luego llama a checkIP para poder validar ambas direcciones y lanzará una excepción que llevara un mensaje descriptivo si una de las IPs es inválida.

```
45  int main() {
46      try {
47          std::string src, dest;
48
49          src = "212.112.212.11";
50          dest = "212.112.212.12";
51          IPHeader header1(src, dest);
52          header1.print();
53
54          src = "212.112.212.333";
55          dest = "212.112.212.33";
56          IPHeader header2(src, dest);
57          header2.print();
58      }
59      catch (const std::invalid_argument& e) {
60          std::cout << e.what() << std::endl;
61      }
62
63      return 0;
64  }
```

Finalmente con la Int Main, este ayudará con crear dos objetos IPHeader dentro de un bloque try-catch para ambas de las IPs si son válidas. Y después el bloque catch se encargará de capturar la excepción y mostrar un mensaje de error si la IP es inválida.

Con todo eso en mente, este seria el resultado final de este ejemplo de Cisco:



TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

EJEMPLO 9: “Exceptions: using in program”

Para esta última actividad de cisco, lo primero que hay que hacer es crear una clase contenedor que implementa una versión simplificada del famoso juego de la Torre de Hanói, utilizando una clase contenedora (Towers) que maneja tres torres (vectores) y tres discos de distintos tamaños (3, 2 y 1).

```cpp
#include <iostream>
#include <vector>
#include <stdexcept>
#include <sstream>

using namespace std;

class InvalidTowerException : public exception {
public:
    const char* what() const noexcept override {
        return "Error: torre fuera del rango (1-3).";
    }
};

class InvalidMoveException : public exception {
public:
    const char* what() const noexcept override {
        return "Error: no se puede colocar un disco más grande sobre uno más pequeño.";
    }
};

class EmptyTowerException : public exception {
public:
    const char* what() const noexcept override {
        return "Error: la torre de origen está vacía.";
    }
};
```

Primero se crearan tres diferentes clases (las clases “InvalidTowerException”, “InvalidMoveException”, y “EmptyTowerException”), las cuales se encargaran de hacer lo siguiente:

- En el caso de “InvalidTowerException”, éste se encargará de lanzar cuando se intenta mover desde o hacia una torre inexistente (distinta de 1, 2 o 3).
- En el caso de “InvalidMoveException”, éste se encargará de lanzar si se intenta mover un disco desde una torre vacía.
- En el caso de “EmptyTowerException”, éste se encargará de lanzar si se intenta poner un disco grande encima de uno más pequeño.

Cada una de estas clases ayuda a redefinir el método what() para mostrar un mensaje claro de error.

TERCER REPORTE DE PROGRAMACIÓN ORIENTADA A OBJETOS

```cpp
class Towers {
private:
    vector<vector<int>> towers;

public:
    Towers() {
        towers.resize(3);
        towers[0] = { 3, 2, 1 };
    }

    void move(int from, int to) {
        if (from < 1 || from > 3 || to < 1 || to > 3)
            throw InvalidTowerException();

        if (from == to)
            return;

        from--;
        to--;

        if (towers[from].empty())
            throw EmptyTowerException();

        int disk = towers[from].back();

        if (!towers[to].empty() && towers[to].back() < disk)
            throw InvalidMoveException();

        towers[from].pop_back();
        towers[to].push_back(disk);
    }
```

Después se creará la clase “Towers”, la cual se encargará de representar el estado del juego donde utilizara un vector con tres vectores internos los cuales representan una torre con sus discos.

No solamente eso, también implementará un constructor que se encargará de inicializar tres torres.

```cpp
int main() {
    Towers game;
    int moveCode;

    while (true) {
        cin >> moveCode;
        if (moveCode == 0)
            break;

        int from = moveCode / 10;
        int to = moveCode % 10;

        try {
            game.move(from, to);
        }
        catch (const exception& e) {
            cout << e.what() << endl;
        }
    }

    game.print();
    return 0;
}
```

Finalmente con la Int Main, este se encargará de controlar la ejecución del juego, donde creará el objeto Towers game y pedirá los movimientos (o coordenadas) en formato XY, a lado de hacer que llame a game.move(from, to) dentro de un bloque try-catch para capturar errores y al final, muestra el estado final con game.print().

Con todo eso en mente, el resultado final de este último ejercicio de Cisco Módulo 3 seria lo siguiente: