

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Laboratorio de Programación de Sistemas				
TÍTULO DE LA PRÁCTICA:	Excepciones en C++				
NÚMERO DE PRÁCTICA:	Guía 7	AÑO LECTIVO:	2023	NRO. SEMESTRE:	V
FECHA DE PRESENTACIÓN	16/07/2023	HORA DE PRESENTACIÓN	22:55		
INTEGRANTE (s) <i>Calcina Puma Esteven Antonio</i> <i>Galvez Quilla Henry Isaias</i> <i>Paredes Quispe Jose Andre</i>				NOTA (0-20)	
DOCENTE(s): Mg. Edith Giovanna Cano Mamani					

RESULTADOS Y PRUEBAS
I. EJERCICIOS RESUELTOS: EJERCICIO 01 Crear un programa donde exista una función que realice la división de dos números que se le pasan como parámetros y devuelva el resultado. En el caso de que el divisor sea cero se tendrá que generar una excepción que será capturada en la función main.

```
1 #include <iostream>
2
3 using namespace std;
4
5 // Función que realiza la división de dos números y lanza una excepción si el divisor es cero
6 int division(int dividendo, int divisor) {
7     if (divisor == 0) {
8         throw "El divisor es cero"; // Lanzar excepción de tipo const char* con mensaje de error
9     }
10
11     return dividendo / divisor;
12 }
13
14 int main() {
15     int dividendo, divisor;
16
17     cout << "Ingresa el dividendo: ";
18     cin >> dividendo;
19
20     cout << "Ingresa el divisor: ";
21     cin >> divisor;
22
23     try {
24         int resultado = division(dividendo, divisor); // Llamar a la función de división
25         cout << "El resultado de la división es: " << resultado; // Mostrar el resultado si no hay excepción
26     } catch (const char *excepcion) { // Capturar la excepción de tipo const char*
27         cout << excepcion; // Mostrar el mensaje de error de la excepción capturada
28     }
29
30     return 0;
31 }
```

```
Ingresa el dividendo: 8
Ingresa el divisor: 2
El resultado de la división es: 4
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

```
Ingresa el dividendo: 6
Ingresa el divisor: 0
El divisor es cero
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

EJERCICIO 02

Modificar el ejercicio anterior de la división por 0 para que ahora la excepción generada dentro de la función que realiza la división se trate ahí y luego la relance para que sea tratada en el main también.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int division(int dividendo, int divisor) {
6     try {
7         if (divisor == 0) {
8             throw "El divisor es cero"; // Lanzar excepción de tipo const char* con mensaje de error
9         }
10
11         return dividendo / divisor;
12     } catch (const char *excepcion) { // Capturar la excepción de tipo const char*
13         cout << excepcion; // Mostrar el mensaje de error de la excepción capturada
14         throw; // Relanzar la excepción para que sea tratada en el bloque catch del main()
15     }
16 }
17
18 int main() {
19     int dividendo, divisor;
20
21     cout << "Ingresa el dividendo: ";
22     cin >> dividendo;
23
24     cout << "Ingresa el divisor: ";
25     cin >> divisor;
26
27     try {
28         int resultado = division(dividendo, divisor); // Llamar a la función de división
29         cout << "El resultado de la división es: " << resultado; // Mostrar el resultado si no hay excepción
30     } catch (const char *excepcion) { // Capturar la excepción de tipo const char*
31         cout << excepcion; // Mostrar el mensaje de error de la excepción capturada
32     }
33
34     return 0;
35 }
```

```
Ingresa el dividendo: 12
Ingresa el divisor: 4
El resultado de la división es: 3
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

```
Ingresa el dividendo: 11
Ingresa el divisor: 0
El divisor es ceroEl divisor es cero
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

3. Para los siguientes casos, proponer y explicar dos ejemplos en C++ considerando:

a) Excepciones provocadas por asignación de memoria insuficiente (generadas por el operador new).

```
7 #include <iostream>
6 using namespace std;
5
4 int main(){
3     int *x;
2     long int y=100000000000;
1     try{
0         x=new int[y];
9         x[0]=10;
8         //convierte a x en un Puntero de tipo void
7         cout<<"Puntero: "<<(void *)x<<endl;
6         delete[] x;
5     }
4     catch(bad_alloc& e){
3         cout<<"Memoria insuficiente"<<
2         e.what()<<endl;
1     }
0     return 0;
1 }
```

```
Memoria insuficientestd::bad_alloc
stv@stv--pc: [~/PS/lab07]$
```

b)Excepciones por tipos de datos incorrectos (ejemplo, al solicitar un dato numérico el usuario digita letras).



```
13 #include <iostream>
12 #include<string>
11 using namespace std;
10
9 void error(string s){
8     throw runtime_error(s);
7 }
6 double some_function(){
5     double d=0;
4     cin>>d;
3     if(!cin) error("No se pudo leer dato de tipo double");
2     return d;
1
4 }
```

```
1 int main(){
2     try{
3         some_function();
4     }
5     catch(runtime_error& e){
6         cout<<e.what()<<endl;
7     }
8 }
9 return 0;
10 }
```

```
stv@stv--pc: [~/PS/lab07]$ ./b
a
No se pudo leer dato de tipo double
stv@stv--pc: [~/PS/lab07]$
```

4. Considerar una función raíces que calcula las raíces cuadradas de una ecuación cuadrática. Diseñar una función de modo que se lancen excepciones si no existen raíces reales o el primer coeficiente es cero. El tipo de excepción es error y los valores serán No_raíces_reales y primer_coeficiente_cero.

```
1  #include <iostream>
2  #include <cmath>
3
4  using namespace std;
5
6  // Clase para representar excepciones
7  class Error {
8  public:
9      Error(string tipo, string valor) : tipo(tipo), valor(valor) {}
10
11     string getTipo() { return tipo; }
12
13     string getValor() { return valor; }
14
15 private:
16     string tipo;
17     string valor;
18 };
19
20 // Función que calcula las raíces de una ecuación cuadrática y lanza excepciones en casos específicos
21 void raices(double a, double b, double c, double &x1, double &x2) {
22     if (a == 0) {
23         throw Error("primer_coeficiente_cero", "a"); // Lanzar excepción cuando el primer coeficiente es cero
24     }
25
26     double discriminante = b * b - 4 * a * c;
27     if (discriminante < 0) {
28         throw Error("No_raices_reales", ""); // Lanzar excepción cuando no existen raíces reales
29     }
30
31     x1 = (-b + sqrt(discriminante)) / (2 * a); // Calcular la primera raíz
32     x2 = (-b - sqrt(discriminante)) / (2 * a); // Calcular la segunda raíz
33 }
34
35 int main() {
36     double a, b, c, x1, x2;
37     cout << "ax^2+bx+c=0" << endl;
38     cout << "Ingresa el valor de a: ";
39     cin >> a;
40
41     cout << "Ingresa el valor de b: ";
42     cin >> b;
43
44     cout << "Ingresa el valor de c: ";
45     cin >> c;
46
47     try {
48         raices(a, b, c, x1, x2); // Calcular las raíces utilizando la función raices
49         cout << "Las raíces de la ecuación son x1 = " << x1 << " y x2 = " << x2 << endl;
50     } catch (Error e) {
51         // Mostrar información de la excepción capturada
52         cout << "Ocurrió un error: " << e.getTipo() << ", " << e.getValor() << endl;
53     }
54
55     return 0;
56 }
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 7</p>

```
ax^2+bx+c=0
Ingresa el valor de a: 0
Ingresa el valor de b: 2
Ingresa el valor de c: 1
Ocurrió un error: primer_coeficiente_cero, a

...Program finished with exit code 0
Press ENTER to exit console.
```

```
ax^2+bx+c=0
Ingresa el valor de a: 2
Ingresa el valor de b: 4
Ingresa el valor de c: 4
Ocurrió un error: No_raices_reales,

...Program finished with exit code 0
Press ENTER to exit console.
```

```
ax^2+bx+c=0
Ingresa el valor de a: 1
Ingresa el valor de b: 2
Ingresa el valor de c: 1
Las raíces de la ecuación son x1 = -1 y x2 = -1

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Crear una jerarquía de clases de excepciones que permitan controlar los posibles errores que surjan de una manera controlada, clara y elegante. Para ello, crear una clase padre/base

ComputerException

que tendrá solamente un campo de tipo string message y que todas las demás clases de excepciones heredarán de ella. Ahora, crear las clases InputException, ProcessorException y OutputException que manejarán los respectivos errores a esos dispositivos. Por ejemplo, si se tiene un tipo Keyboard que es un fichero del que se van a leer las entradas, KeyboardException manejará los posibles problemas que pudieran aparecer a la hora de abrir el fichero, leerlo, etc.

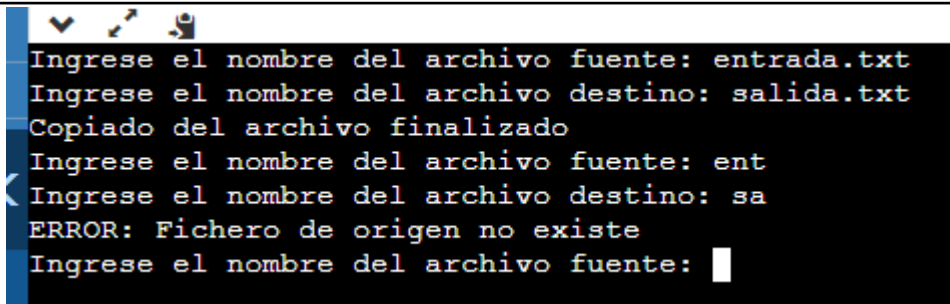
```
main.cpp  entrada.txt  salida.txt  Ctrl+S
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4
5  using namespace std;
6
7  class ComputerException : public exception {
8  public:
9      ComputerException() : exception() {}
10     ComputerException(const char* msg) {message = msg;}
11     const char* what() const throw() {
12         return message;
13     }
14     protected:
15         const char* message;
16 };
17
18 class InputException : public ComputerException {
19 public:
20     InputException() : ComputerException("ERROR: Fichero de origen no existe") {}
21 };
22
23 class ProcessorException : public ComputerException {
24 public:
25     ProcessorException() : ComputerException("Copiado del archivo finalizado") {}
26 };
27
28 class OutputException : public ComputerException {
29 public:
30     OutputException() : ComputerException("ERROR: Fichero de origen no se puede abrir") {}
31 };
32
```



```
33
34 void CopiaFichero(string Origen, string Destino) {
35     unsigned char buffer[1024];
36     int leido;
37
38     ifstream fe(Origen, ios::in | ios::binary);
39     if(!fe.good()) throw InputException();
40
41     ofstream fs(Destino, ios::out | ios::binary);
42     if(!fs.good()) throw OutputException();
43
44     do {
45         fe.read(reinterpret_cast<char *> (buffer), 1024);
46         leido = fe.gcount();
47         fs.write(reinterpret_cast<char *> (buffer), leido);
48     } while(leido);
49
50     fe.close();
51     fs.close();
52
53     throw ProcessorException();
54 }
```

```
55
56 int main() {
57     string Desde;
58     string Hacia;
59
60     while (true) {
61         try {
62             cout << "Ingrese el nombre del archivo fuente: ";
63             cin >> Desde;
64             cout << "Ingrese el nombre del archivo destino: ";
65             cin >> Hacia;
66             CopiaFichero(Desde, Hacia);
67             break;
68         } catch(ComputerException &ex) {
69             cout << ex.what() << endl;
70         }
71     }
72     return 0;
73 }
74
```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 10



```

Ingrese el nombre del archivo fuente: entrada.txt
Ingrese el nombre del archivo destino: salida.txt
Copiado del archivo finalizado
Ingrese el nombre del archivo fuente: ent
Ingrese el nombre del archivo destino: sa
ERROR: Fichero de origen no existe
Ingrese el nombre del archivo fuente: 

```

II. PRUEBAS

¿Con que valores comprobaste que tu práctica estuviera correcta? ¿Qué resultado esperabas obtener para cada valor de entrada? ¿Qué valor o comportamiento obtuviste para cada valor de entrada?

Cada ejercicio se probó con los ficheros generados y se comprobaba con las indicaciones de cada uno de los ejercicios

III. CUESTIONARIO:

1) ¿C++ es el único lenguaje de programación que trata excepciones?

No, C++ no es el único lenguaje de programación que maneja excepciones. Otros lenguajes de programación que manejan excepciones incluyen Java, C#, Python y PHP.

2) ¿Cuál es la ventaja de manejar excepciones?

Hay varias ventajas de manejar excepciones. Las excepciones pueden ayudar a proteger tu código de errores y a mejorar la seguridad del mismo. También pueden ayudar a hacer que tu código sea más fácil de leer y comprender.

CONCLUSIONES

Hemos llegado a las siguientes conclusiones:

- El manejo de excepciones puede mejorar la legibilidad y mantenibilidad del código al separar el código de la lógica principal del código de manejo de errores. Esto facilita la lectura y comprensión del programa, y también ayuda a mantener el código más limpio y organizado.
- Las excepciones permiten definir flujos de control específicos para el manejo de errores, evitando la propagación descontrolada de errores a través de las llamadas de funciones y permitiendo centralizar el tratamiento de errores en un solo lugar.
- El uso de excepciones permite separar las preocupaciones entre el código funcional y el manejo de errores. Esto facilita la modificación y extensión del código sin afectar el mecanismo de manejo de errores.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 11</p>

- Las excepciones proporcionan información más significativa sobre el error ocurrido, lo que simplifica la identificación y resolución de problemas. Esto evita el uso de códigos de error poco claros, como códigos de error numéricos o constantes.
- Las excepciones pueden no ser adecuadas para todos los escenarios. En ciertos entornos con restricciones de recursos muy estrictas, el uso intensivo de excepciones puede afectar el rendimiento.
- Las excepciones permiten liberar automáticamente los recursos adquiridos, como memoria o archivos, mediante el uso de constructores y destructores especiales. Esto garantiza una limpieza adecuada en caso de errores.
- Es importante utilizar excepciones de manera adecuada y lanzarlas solo en situaciones realmente excepcionales. Lanzar excepciones innecesarias o en exceso puede llevar a un código confuso y difícil de depurar.
- Capturar excepciones también debe hacerse con cuidado y consideración. Capturar excepciones demasiado genéricas o ignorarlas por completo puede ocultar problemas en el código y dificultar la depuración.

METODOLOGÍA DE TRABAJO

Se siguieron los siguientes pasos:

- Comprensión del problema
- Identificación de la entrada y salida
- Diseño del algoritmo
- Codificación
- Depuración
- Pruebas
- Optimización
- Entrega o implementación

REFERENCIAS Y BIBLIOGRAFÍA

- [1] P.J. Deitel and H.M. Deitel, "Cómo Programar en C++", México, Ed. Pearson Educación, 2009
- [2] B. Stroustrup, "El Lenguaje de Programación C++", Madrid, Adisson Pearson Educación, 2002
- [3] B. Eckel, "Thinking in C++", Prentice Hall, 2000