



Error Handling. Exception

VGP131-OOP II

Instructor: Ivaldo Tributino



Exception Handling

- A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**.^[2]

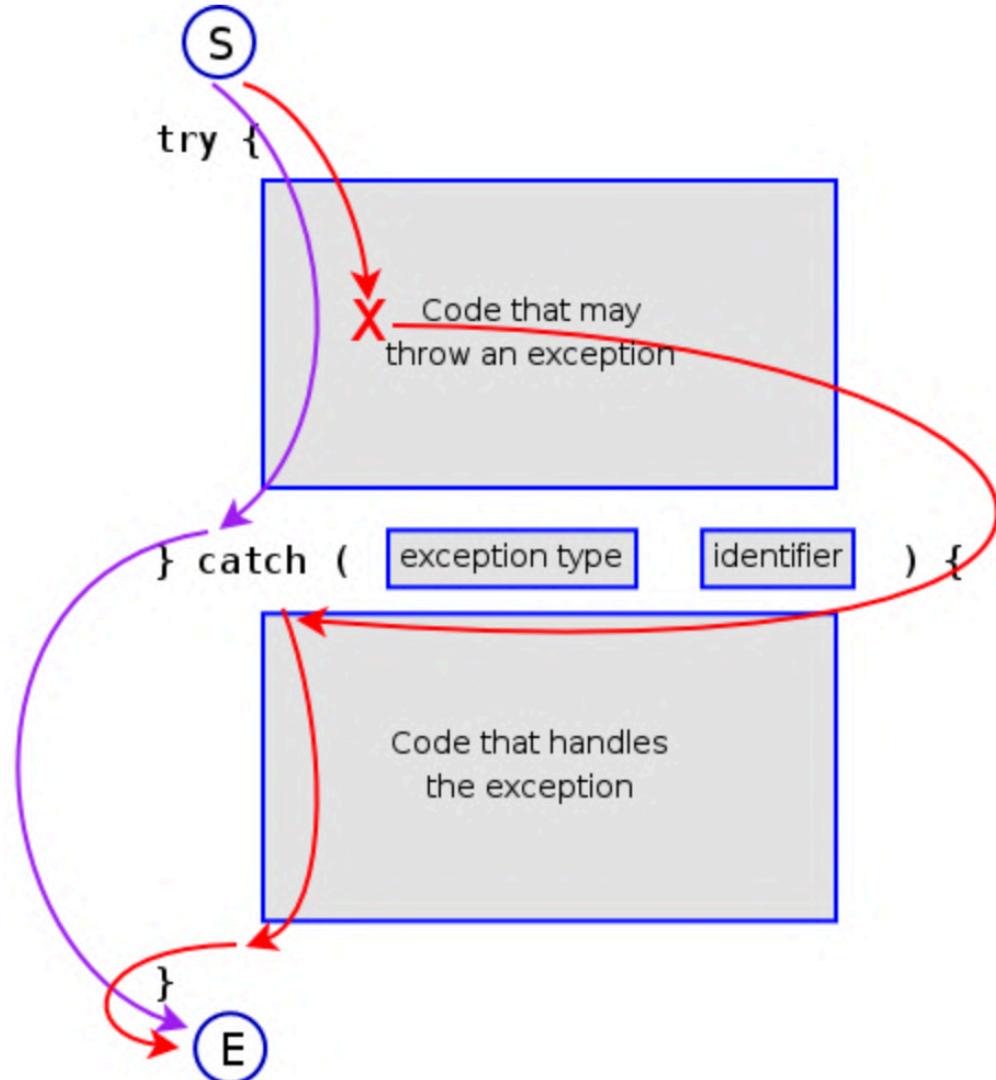


Figure from : stackoverflow.com

Try, throw and catch.

Try

A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Throw

A program throws an exception when a problem shows up. This is done using a throw keyword.

Catch

A program catches an exception with an exception handler at the place in a program where you want to handle the problem.

Throwing and Catching Exceptions

Throwing Exceptions

Exceptions can be thrown anywhere within a code block using **throw** statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

Catching Exceptions

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

Throwing and Catching Exceptions

The screenshot shows a code editor interface with a sidebar containing various icons. The main area displays a C++ file named `main.cpp`. The code implements exception handling for division operations.

```
main.cpp
68
69  double division(int a, int b, char c) {
70      const char& model = c;
71      if( b == 0 && model == 'a') {
72          throw "Division by zero condition!";
73      }
74      else if( b == 0 && model == 'b'){
75          throw INT_MAX;
76      }
77      return (a/(double)b);
78  }
79
80 int main(){
81
82 //***** Throwing and Catching Exceptions *****
83 // ---- Throwing and Catching Exceptions -----
84 //***** Throwing and Catching Exceptions *****
85 cout << "---- Throwing and Catching Exceptions ----" << '\n';
86
87 try {
88     cout << division(4,3, 'b') << endl;
89     cout << division(4,0, 'b') << endl;
90 }
91 catch (char const* msg)
92 {
93     cerr << msg << '\n'; // cerr is the standard error stream which is used to output the errors.
94 }
95 catch (const int& value)
96 {
97     cerr << value << '\n';
98 }
99
100
101
```

The code defines a function `division` that performs integer division. It handles two specific cases where the divisor is zero: if the dividend is also zero, it throws a string message; if the dividend is non-zero, it throws the constant `INT_MAX`. The `main` function demonstrates this by printing the results of two division operations to the standard output and catching the exceptions to print them to the standard error stream (`cerr`).

The terminal output window shows the execution of the program:

```
---- Throwing and Catching Exceptions -----
1.33333
2147483647
(base) Ivaldo:Week9 admin$
```

The status bar at the bottom indicates the current file is `main.cpp`, the master branch, and the code has 101 lines and 4 spaces.

Throwing and Catching Exceptions

The screenshot shows a code editor interface with a dark theme. On the left, there's a vertical toolbar with various icons: a file icon, a search icon, a '10K+' icon, a zoom icon, a copy/paste icon, a GitHub icon, and a refresh/circular arrow icon. Below the toolbar is a status bar with a gear icon and the number '1'.

The main area displays a C++ program named `main.cpp`. The code implements exception handling for division operations. It includes a function `division` that throws exceptions for division by zero and invalid inputs. The `main` function demonstrates throwing and catching these exceptions, outputting results to the standard error stream (`cerr`).

```
main.cpp
68
69  double division(int a, int b, char c) {
70      const char& model = c;
71      if( b == 0 && model == 'a') {
72          throw "Division by zero condition!";
73      }
74      else if( b == 0 && model == 'b'){
75          throw INT_MAX;
76      }
77      return (a/(double)b);
78  }
79
80 int main(){
81
82     //*****----- Throwing and Catching Exceptions -----
83     // ----- Throwing and Catching Exceptions -----
84     //*****----- Throwing and Catching Exceptions -----
85     cout << "----- Throwing and Catching Exceptions -----" << '\n';
86
87     try {
88         cout << division(4,3, 'b') << endl;
89         cout << division(4,0, 'a') << endl;
90     }
91     catch (char const* msg)
92     {
93         cerr << msg << '\n'; // cerr is the standard error stream which is used to output the errors.
94     }
95     catch (const int& value)
96     {
97         cerr << value << '\n';
98     }
99
100
101
```

At the bottom of the editor, there are tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE. The TERMINAL tab is currently selected, showing the command-line output:

```
---- Throwing and Catching Exceptions ----
1.33333
Division by zero condition!
(base) Ivaldo:Week9 admin$
```

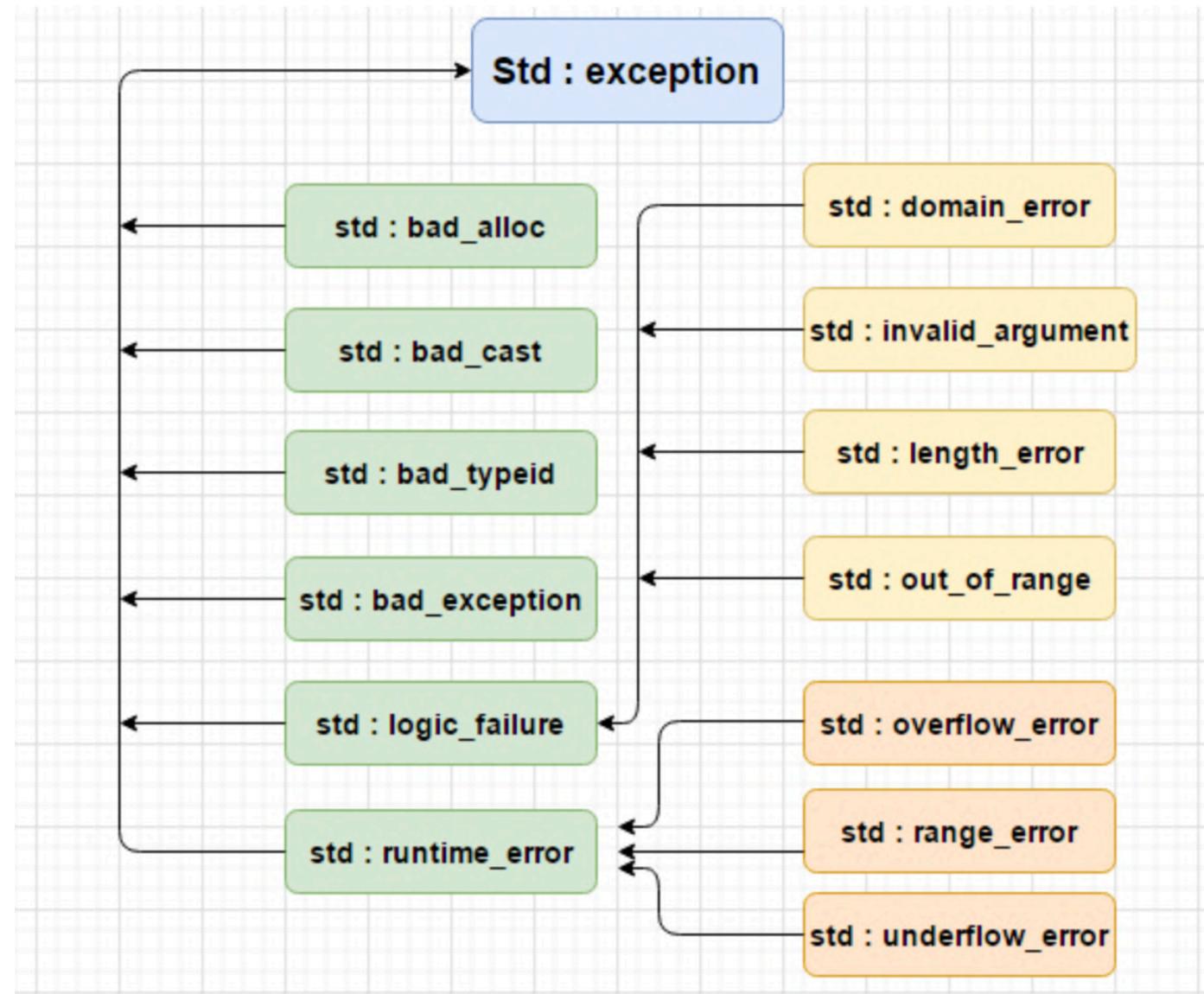
Below the terminal, there are navigation icons for back, forward, and search, along with a 'Live Share' button. The status bar at the bottom right shows the line number (Ln 89), column (Col 33), spaces (Spaces: 4), encoding (UTF-8), file type (LF C++), and a small icon.

catch(...)

- If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows

```
try {  
    // protected code  
} catch(...) {  
    // code to handle any exception  
}
```

Standard Library Exception Hierarchy



Sr.No Exception & Description

1	std::exception An exception and parent class of all the standard C++ exceptions.
2	std::bad_alloc This can be thrown by new .
3	std::bad_cast This can be thrown by dynamic_cast .
4	std::bad_exception This is useful device to handle unexpected exceptions in a C++ program.
5	std::bad_typeid This can be thrown by typeid .
6	std::logic_error An exception that theoretically can be detected by reading the code.
7	std::domain_error This is an exception thrown when a mathematically invalid domain is used.
8	std::invalid_argument This is thrown due to invalid arguments.
9	std::length_error This is thrown when a too big std::string is created.
10	std::out_of_range This can be thrown by the 'at' method, for example a std::vector and std::bitset<>::operator[]().
11	std::runtime_error An exception that theoretically cannot be detected by reading the code.
12	std::overflow_error This is thrown if a mathematical overflow occurs.
13	std::range_error This is occurred when you try to store a value which is out of range.
14	std::underflow_error This is thrown if a mathematical underflow occurs.

Std : exception

The screenshot shows a code editor interface with several tabs at the top: main.cpp, calculator.h, test.h, Polygon.cpp, libraries.h, and Polygon.h. The main.cpp tab is active, displaying the following C++ code:

```
95     catch (char const* msg)
96     {
97         cerr << msg << '\n'; // cerr is the standard error stream which is used to output the errors.
98     }
99     catch (const int& value)
100    {
101        cerr << value << '\n';
102    }
103
104 //*****
105 //      ---- std::exception ----
106 //*****
107 cout << "---- std::exception ----" << '\n';
108 S* p = nullptr;
109 try {
110     // An exception of this type is thrown when a typeid operator is applied
111     // to a dereferenced null pointer value of a polymorphic type.
112     std::cout << typeid(*p).name() << '\n';
113 }
114 catch (const std::bad_alloc& e)
115 {
116     std::cerr << e.what() << '\n';
117 }
118 catch(const std::bad_typeid& e) {
119     std::cerr << e.what() << '\n';
120 }
121
122
123
124
125
126
127
128
129
```

A callout box highlights the following code snippet:

```
struct S { // The type has to be polymorphic
    virtual void f();
};
```

The bottom of the editor shows the terminal output:

```
---- std::exception ----
std::bad_typeid
(base) Ivaldo:Week9 admin$
```

The bottom navigation bar includes links for PROBLEMS, OUTPUT, TERMINAL (which is underlined), and DEBUG CONSOLE. On the far right, there are icons for bash, terminal, file operations, and other tools.

std::exception::what

Pointer to a null-terminated string with explanatory information. The pointer is guaranteed to be valid at least until the exception object from which it is obtained is destroyed, or until a non-const member function on the exception object is called.

What



The screenshot shows a terminal window with the following output:

```
----- std::exception::what -----  
{0:1}  
{1:1}  
{2:1}  
{3:2}  
{4:3}  
{5:5}  
(base) Ivaldo:Week9 admin$
```

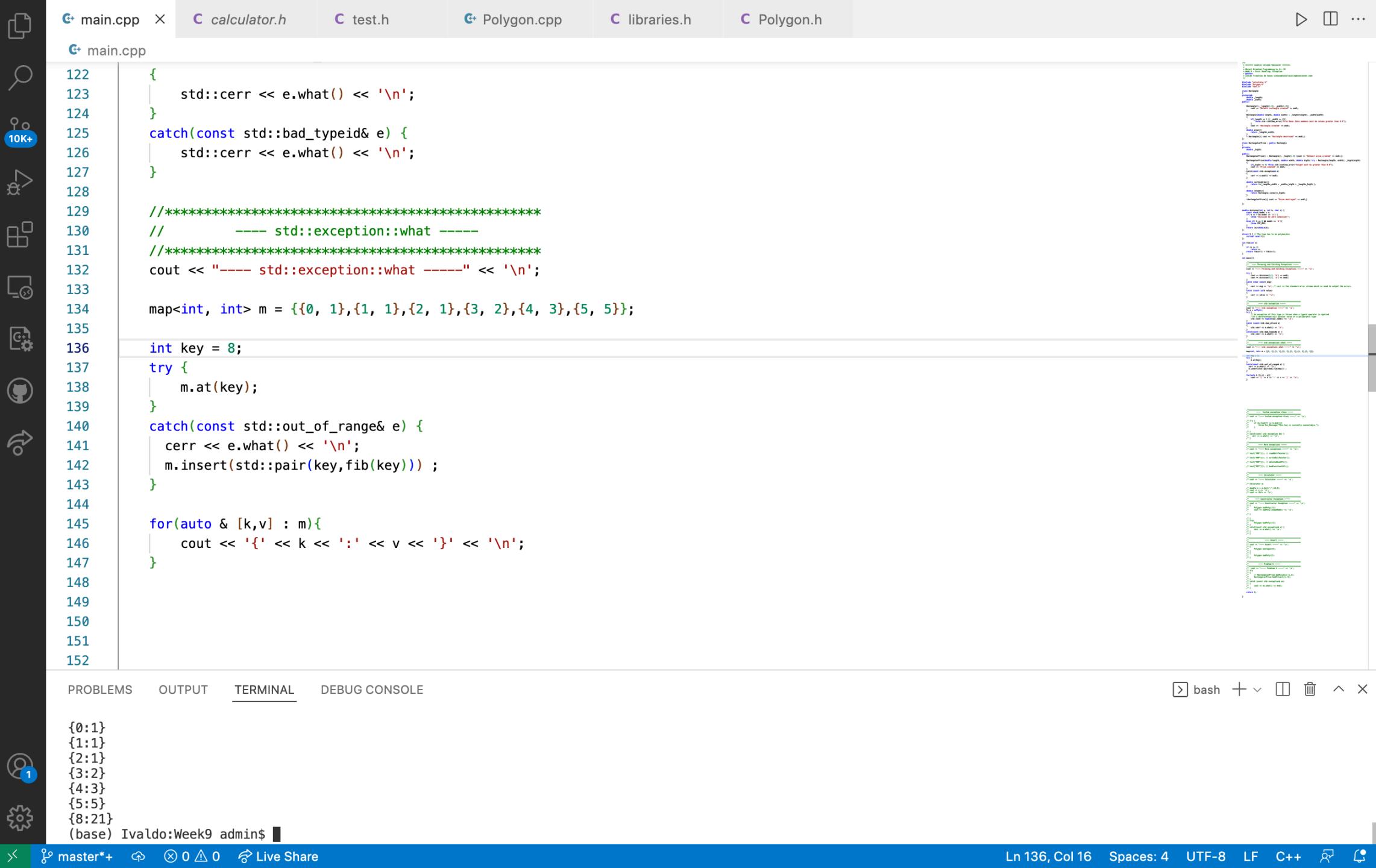
At the bottom of the terminal window, there are icons for bash, a plus sign, a minus sign, a square, a trash can, and a close button.

The main code editor area displays the following C++ code:

```
122     {  
123         std::cerr << e.what() << '\n';  
124     }  
125     catch(const std::bad_typeid& e) {  
126         std::cerr << e.what() << '\n';  
127     }  
128  
129     //*****  
130     //      ---- std::exception::what ----  
131     //*****  
132     cout << "---- std::exception::what ----" << '\n';  
133  
134     map<int, int> m = {{0, 1},{1, 1},{2, 1},{3, 2},{4, 3},{5, 5}};  
135  
136     int key = 5;  
137     try {  
138         m.at(key);  
139     }  
140     catch(const std::out_of_range& e) {  
141         cerr << e.what() << '\n';  
142         m.insert(std::pair(key,fib(key)));  
143     }  
144  
145     for(auto & [k,v] : m){  
146         cout << '{' << k << ':' << v << '}' << '\n';  
147     }  
148  
149  
150  
151  
152
```

The sidebar on the left contains various icons for file operations, search, and navigation.

What



The screenshot shows a code editor interface with the following details:

- File Tabs:** main.cpp, calculator.h, test.h, Polygon.cpp, libraries.h, Polygon.h.
- Sidebar:** Includes icons for file operations (copy, paste, move), search, and a "10K+" button.
- Code Area:** Displays the following C++ code:

```
122     {
123         std::cerr << e.what() << '\n';
124     }
125     catch(const std::bad_typeid& e) {
126         std::cerr << e.what() << '\n';
127     }
128
129 //*****
130 //      ---- std::exception::what -----
131 //*****
132 cout << "---- std::exception::what ----" << '\n';
133
134 map<int, int> m = {{0, 1},{1, 1},{2, 1},{3, 2},{4, 3},{5, 5}};
135
136 int key = 8;
137 try {
138     m.at(key);
139 }
140 catch(const std::out_of_range& e) {
141     cerr << e.what() << '\n';
142     m.insert(std::pair(key,fib(key)));
143 }
144
145 for(auto & [k,v] : m){
146     cout << '{' << k << ':' << v << '}' << '\n';
147 }
148
149
150
151
152 }
```
- Bottom Navigation:** PROBLEMS, OUTPUT, TERMINAL (underlined), DEBUG CONSOLE.
- Terminal Output:** Shows the output of the program:

```
{0:1}
{1:1}
{2:1}
{3:2}
{4:3}
{5:5}
{8:21}
```
- Bottom Status Bar:** master*+, Live Share, Ln 136, Col 16, Spaces: 4, UTF-8, LF, C++, and icons for file operations.

Custom exception class

```
class Exc_Message : public std::exception{
private:
    std::string message_;
public:
    Exc_Message(const std::string& message): message_(message){}
    const char* what() const noexcept override {
        return message_.c_str();
    }
};
```

Custom Messages

The screenshot shows a code editor interface with several tabs at the top: main.cpp, test.h, Polygon.cpp, libraries.h, and Polygon.h. The main.cpp tab is active, displaying C++ code. The code includes a standard exception handling section and a custom exception class section. The custom exception class handles a key that is currently unavailable.

```
141     cerr << e.what() << '\n';
142     m.insert(std::pair(key,fib(key))) ;
143 }
144
145 for(auto & [k,v] : m){
146     cout << '{' << k << ':' << v << '}' << '\n';
147 }
148
149 // ****
150 // ----- Custom exception class -----
151 // ****
152 cout << "----- Custom exception class -----" << '\n';
153
154 try {
155     if (m.find(7) == m.end()){
156         throw Exc_Message("This key is currently unavailable.");
157     }
158 }
159 catch(const std::exception &e) {
160     cerr << e.what() << '\n';
161 }
```

The code editor's sidebar on the left contains various icons for file operations, search, and navigation. The bottom left corner shows a circular icon with a '1', indicating one notification or error. The bottom right corner shows a terminal interface with the following output:

```
---- std::exception::what ----
map::at: key not found
{0:1}
{1:1}
{2:1}
{3:2}
{4:3}
{5:5}
{8:21}
----- Custom exception class -----
This key is currently unavailable.
(base) Ivaldo:Week9 admin$
```

The bottom status bar indicates the current line (Ln 170), column (Col 1), and other settings like spaces (4), encoding (UTF-8), and file type (LF C++).

Calling test functions from a map

```
map<string, std::function<void()>> test
{
    { "RNP", readNullPointer },
    { "WNP", writeNullPointer },
    { "DWP", deletedWeakPtr},
    { "BFC", badFunctionCall}
};
```

Test

```
main.cpp test.h libraries.h Polygon.h Polygon.cpp
```

```
C test.h > readNullPointer()
11 #include "libraries.h"
12
13 class Exc_Message : public std::exception{
14 private:
15     std::string message_;
16 public:
17     Exc_Message(const std::string& message): message_(message){};
18     const char* what() const noexcept override {
19         return message_.c_str();
20     }
21 };
22
23
24 void readNullPointer()
25 {
26     int* p = nullptr;
27     try
28     {
29         if (!p){
30             throw Exc_Message("Read from nullptr");
31         }
32         cout << *p << endl;
33     }
34     catch (const std::exception& e)
35     {
36         cerr << e.what() << endl;
37     }
38 }
39
40 void writeNullPointer()
41 {
42     int* p = nullptr;
43     try
44     {
45         if (!p){
46             throw Exc_Message("Write to nullptr");
47         }
48         *p = 42;
49     }
50     catch (const std::exception& e)
```

Ln 25, Col 2 Spaces: 4 UTF-8 LF C++ Mac Live Share

Test

The screenshot shows a code editor interface with several tabs at the top: main.cpp, test.h, libraries.h, Polygon.h, and Polygon.cpp. The current file is test.h, which contains the following code:

```
55
56     void deletedWeakPtr()
57     {
58
59         std::shared_ptr<int> p1(new int(42));
60         std::weak_ptr<int> wp(p1);
61         p1.reset();
62         try {
63             if(wp.expired()){
64                 throw Exc_Message("Refers to an already deleted object");
65                 std::shared_ptr<int> p2(wp);
66             }
67
68         } catch(const std::exception& e) {
69             cerr << e.what() << endl;
70         }
71     }
72
73     void badFunctionCall()
74     {
75
76         std::function<int()> f = nullptr;
77         try {
78             if(!f){
79                 throw Exc_Message("The function wrapper has no target.");
80             }
81
82         } catch(const std::exception& e) {
83             cerr << e.what() << endl;
84         }
85     }
86
87     map<string, std::function<void()>> test
88     {
89         { "RNP", readNullPointer },
90         { "WNP", writeNullPointer },
91         { "DWP", deletedWeakPtr},
92         { "BFC", badFunctionCall}
93     };
94 }
```

The code uses `std::shared_ptr` and `std::weak_ptr` to handle shared ownership of objects. It also demonstrates the use of `std::function` and `std::map` for function pointers and mapping strings to functions.

Test



```
-----  
Read from nullptr  
Write to nullptr  
Refers to an already deleted object  
The function wrapper has no target.  
(base) Ivaldo:Week9 admin$
```

The screenshot shows a code editor interface with several tabs at the top: main.cpp, test.h, libraries.h, Polygon.h, and Polygon.cpp. The main.cpp tab is active, displaying C++ code. The code includes sections for printing key-value pairs, a custom exception class, and various exception handling blocks. It also contains calls to test functions for different types of pointer-related errors: RNP, WNP, DWP, and BFC. Below the code editor, there are tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE. The TERMINAL tab is selected, showing a terminal window with a stack trace for a segmentation fault. The stack trace includes frames for main, __cxa_throw, operator new, and std::bad_alloc. The bottom status bar indicates the current line (Ln 89), column (Col 34), and the number of selected lines (15 selected). The bottom right corner shows icons for bash, terminal, file, and close.

```
main.cpp X test.h libraries.h Polygon.h Polygon.cpp > ▶ 🔍 ⌂ ...  
main.cpp > main()  
67     for (auto& [k, v] : m){  
68         cout << '{' << k << ':' << v << '}' << '\n';  
69     }  
70  
71 //*****  
72 //      ---- Custom exception class ----  
73 //*****  
74 cout << "---- Custom exception class ----" << '\n';  
75  
76 try {  
77     if (m.find(7) == m.end()){  
78         throw Exc_Message("This key is currently unavailable.");  
79     }  
80 }  
81  
82 catch(const std::exception &e) {  
83     cerr << e.what() << '\n';  
84 }  
85  
86 //*****  
87 //      ---- More exceptions ----  
88 //*****  
89 cout << "---- More exceptions ----" << '\n';  
90  
91 test["RNP"](); // readNullPointer();  
92  
93 test["WNP"](); // writeNullPointer();  
94  
95 test["DWP"](); // deletedWeakPtr();  
96  
97 test["BFC"](); // badFunctionCall();  
98  
99  
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE bash + × ⌂ ^ ×  
-----  
Read from nullptr  
Write to nullptr  
Refers to an already deleted object  
The function wrapper has no target.  
(base) Ivaldo:Week9 admin$
```

Ln 89, Col 34 (15 selected) Spaces: 4 UTF-8 LF C++ Mac 🔍 🔍

ios::exceptions

- In C++ iostreams do not throw exceptions by default. We need to use **std::ios::exceptions** to set the exception behavior.

ios::exceptions

The screenshot shows a code editor interface with the following details:

- File Tabs:** main.cpp, calculator.h, Polygon.cpp, libraries.h, Polygon.h.
- Sidebar Icons:** Copy, Find, 10K+, Open, Split, Minimize, GitHub, Refresh.
- Code Area:** The main.cpp file contains C++ code demonstrating exception handling for std::ifstream. It includes comments for ios::exceptions, and handles various exceptions like failbit and badbit.
- Terminal Area:** Shows the command (base) Ivaldo:Week9 admin\$./main followed by the error message "Error: Ifstream failbit or badbit is set.".
- Right Panel:** Shows a detailed stack trace and memory dump for the error, with numerous lines of assembly and memory addresses.
- Bottom Bar:** PROBLEMS, OUTPUT, TERMINAL (underlined), DEBUG CONSOLE, bash, main.
- Status Bar:** Ln 190, Col 29 (5 selected), Spaces: 4, UTF-8, LF, C++, Live Share.

ios::exceptions

The screenshot shows a code editor interface with several tabs at the top: main.cpp, calculator.h, Polygon.cpp, libraries.h, and Polygon.h. The main.cpp tab is active, displaying the following C++ code:

```
187     std::ifstream myfile ("test.txt");
188     if(myfile.is_open()){
189         while (!myfile.eof()) myfile.get();
190         myfile.close();
191     }
192     else if(myfile.fail())
193     {
194         cout << "Error: Ifstream failbit or badbit is set." << endl;
195     }
196
197     std::ifstream file;
198     file.exceptions( std::ifstream::failbit | std::ifstream::badbit );
199     try {
200         file.open ("test.txt");
201         while (!file.eof()) file.get();
202         file.close();
203     }
204     catch(std::exception const& e){
205
206         std::cerr << "Show me the error:" << e.what() << endl;
207     }
208     catch (std::ifstream::failure const& e) {
209         std::cerr << e.what() << endl;
210     }
211     catch(...) {
212         cout << "Please show me something" << endl;
213     }
214
215     cout << "Please show me something" << endl;
216
217
218
219
220
```

On the left side, there is a vertical toolbar with various icons. At the bottom, there are tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE. The TERMINAL tab is selected, showing the following output:

```
Error: Ifstream failbit or badbit is set.
Show me the error:ios_base::clear: unspecified iostream_category error
Please show me something
(base) Ivaldo:Week9 admin$
```

On the right side, there is a large sidebar containing a tree view of project files and a search bar.

std::optional

Do not use exceptions for errors that are expected to occur frequently.

In some cases, it is better to use other tools like std::optional. A common use case for optional is the return value of a function that may fail.

ios::optional

The screenshot shows a code editor interface with several tabs at the top: main.cpp, calculator.h, Polygon.cpp, libraries.h, and Polygon.h. The main.cpp tab is active, displaying the following code:

```
205     catch (std::exception const& e) {
206         std::cerr << "Show me the error:" << e.what() << endl;
207     }
208     catch (std::ifstream::failure const& e) {
209         std::cerr << e.what() << endl;
210     }
211     catch(...) {
212         cout << "Please show me something" << endl;
213     }
214
215     cout << "Please show me something" << endl;
216
217     //*****
218     //      ---- std::optional ----
219     //*****
220     cout << "---- std::optional ----" << '\n';
221
222     cout << (divide(9,4)).value() << '\n';
223     cout << *divide(9,4) << '\n';
224     // cout << divide(9,0).value() << '\n';
225     cout << *divide(9,0) << '\n';
226     cout << (divide(9,0)).value_or(42) << '\n';
227
228
229
230
231
232
233
234
235
236
237
238
```

A callout box highlights the following code block:

```
std::optional<double> divide(int a, int b) {
    if (b != 0) return a/(double)b;
    return {};
}
```

The terminal output below shows the execution of the code:

```
---- std::optional ----
2.25
2.25
1.62597e-260
42
(base) Ivaldo:Week9 admin$
```

The status bar at the bottom indicates: Ln 236, Col 1 Spaces: 4 UTF-8 LF C++ Live Share.

Constructor Exception

Constructors don't have a return type, so it's not possible to use return codes. The best way to signal constructor failure is therefore to throw an exception.

condition ? true case : false case

The screenshot shows a code editor interface with several tabs at the top: main.cpp, Polygon.cpp, libraries.h, and Polygon.h. The main.cpp tab is active, displaying the following code:

```
// ****
178
179
180
181
182
183
184
185
186
187
188
189 // ****
190 // ---- Constructor Exception ----
191 // ****
192 cout << "---- Constructor Exception ----" << '\n';
193 {
194     Polygon badPoly(-1);
195     cout << badPoly.getNumberOfSides() << '\n';
196 }
```

The output terminal below shows the execution results:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
```

```
50
2
Division by zero condition!
inf
---- Constructor Exception ----
Constructor Invoked
4294967295
Polygon destroyed
(base) Ivaldo:Week9 admin$
```

A callout box highlights the constructor definition from Polygon.h:

```
Polygon :: Polygon(unsigned int numberSides)
{
    numberSides_ = (numberSides>2)? numberSides : 3;
    cout << "Constructor Invoked" << endl;
}
```

The status bar at the bottom indicates: Ln 203, Col 1 Spaces: 4 UTF-8 LF C++ Live Share

throw expression



The screenshot shows a code editor interface with several tabs at the top: main.cpp, Polygon.cpp, libraries.h, and Polygon.h. The main.cpp tab is active, displaying the following C++ code:

```
181     cout << "---- Calculator ----" << '\n';
182
183     Calculator a;
184
185     a.Call('*','10',5);
186     a.Call('/','10',5);
187     a.Call('/','10',0);
188
189 // *****
190 //      ---- Constructor Exception ----
191 // *****
192 cout << "---- Constructor Exception ----" << '\n';
193 {
194     //    Polygon badPoly(-1);
195     //    cout << badPoly.getNumberOfSides() << '\n';
196 }
197
198 {
199     try{
200         Polygon badPoly(-1);
201     }
202     catch(const std::exception& e) {
203         cerr << e.what() << '\n';
204     }
205 }
```

A callout box highlights the constructor definition in Polygon.h:

```
Polygon :: Polygon(unsigned int numberSides) : numberSides_(numberSides)
{
    // numberSides_ = (numberSides>2)? numberSides : 3;
    if ((numberSides < 3) || (numberSides > INT_MAX)){
        throw std::runtime_error("Polygon is a geometrical figure with three or more sides.");
    }
    cout << "Constructor Invoked" << endl;
}
```

The terminal window at the bottom shows the execution of the program and its output:

```
---- Constructor Exception ----
Polygon is a geometrical figure with three or more sides.
(base) Ivaldo:Week9 admin$
```

The status bar at the bottom right indicates: Ln 214, Col 1 Spaces: 4 UTF-8 LF C++.

Assertions (abort instead of handling exception)

Assertions help to implement checking of preconditions in programs.

Standard library header <cassert>

This header was originally in the C standard library as <assert.h>.

This header is part of the [error handling](#) library.

Assert

```
main.cpp x test.h Polygon.cpp libraries.h Polygon.h
main.cpp > ...
129
130     //*****
131     //      ---- Assert ----
132     //*****
133     cout << "---- Assert ----" << '\n';
134
135     Polygon pentagon(5);
136
137
138     Polygon badPoly(2);
139
140
141
142     return 0;
143
144 }
145
146
147 Polygon :: Polygon(unsigned int numberSides) : numberSides_(numberSides)
148 {
149     assert(("Polygon is a geometrical figure with three or more sides.", numberSides_ >= 3));
150
151     // numberSides_ = (numberSides>2)? numberSides : 3;
152     // if ((numberSides < 3) || (numberSides > INT_MAX)){
153     //     throw std::runtime_error("Polygon is a geometrical figure with three or more sides.");
154     // }
155     cout << "Constructor Invoked" << endl;
156
157 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash + ×

```
---- Assert ----
Constructor Invoked
Polygon destroyed
Assertion failed: (("Polygon is a geometrical figure with three or more sides.", numberSides_ >= 3)), function Polygon, file Polygon.cpp, line 23.
Abort trap: 6
(base) Ivaldo:Week9 admin$
```



References:

1. Tutorialspoint (2022/02/25) C++ Exception Handling,
https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm
2. Cristian Adam's blog (2022/02/25) C++ NullPointerException in C++,
<https://cristianadam.eu/20160914/nullpointerexception-in-c-plus-plus/>
3. Isocpp (2022/02/25) Exceptions and Error Handling, <https://isocpp.org/wiki/faq/exceptions#ctors-can-throw>

Standard error stream (cerr): **cerr** is the standard error stream which is used to output the errors.

It is an instance of the [ostream class](#). As cerr stream is un-buffered so it is used when we need to display the error message immediately and does not store the error message to display later.

The object of class [ostream](#) that represents the standard error stream oriented to narrow characters(of type char). It corresponds to the C stream [stderr](#).

The “c” in **cerr** refers to “character” and ‘err’ means “error”, Hence cerr means “character error”. It is always a good practice to use **cerr** to display errors.

cerr – Standard Error Stream Object in C++

(source:

<https://www.geeksforgeeks.org/cerr-standard-error-stream-object-in-cpp/>)

Appendix

- ❖ Libraries.h
- ❖ test.h
- ❖ Polygon.h
- ❖ Polygon.cpp
- ❖ main.cpp