Error Handling. Exception

VGP131-O0P 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]

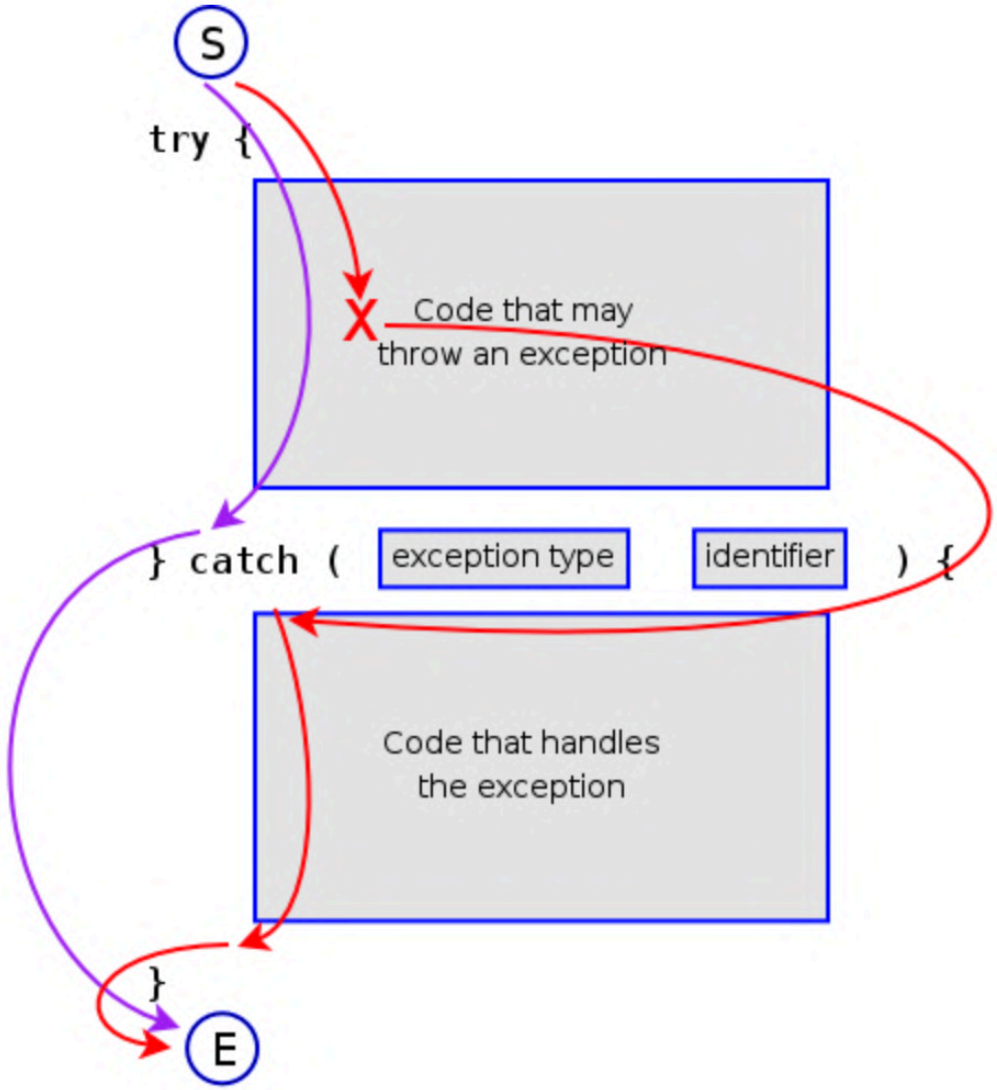


Figure from : stackoverflow.com

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

main.cpp ● | calculator.h | test.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```cpp
double division(int a, int b, char c) {
    const char& model = c;
    if( b == 0 && model == 'a') {
        throw "Division by zero condition!";
    }
    else if( b == 0 && model == 'b'){
        throw INT_MAX;
    }
    return (a/(double)b);
}

int main(){

    //************************************************
    //    ---- Throwing and Catching Exceptions -----
    //************************************************
    cout << "---- Throwing and Catching Exceptions -----" << '\n';

    try {
        cout << division(4,3, 'b') << endl;
        cout << division(4,0, 'b') << endl;
    }
    catch (char const* msg)
    {
        cerr << msg << '\n'; // cerr is the standard error stream which is used to output the errors.
    }
    catch (const int& value)
    {
        cerr << value << '\n';
    }


```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ ⊗ 0 ⚠ 0 Live Share Ln 101, Col 1 Spaces: 4 UTF-8 LF C++

Throwing and Catching Exceptions

main.cpp | calculator.h | test.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```cpp

double division(int a, int b, char c) {
    const char& model = c;
    if( b == 0 && model == 'a') {
        throw "Division by zero condition!";
    }
    else if( b == 0 && model == 'b'){
        throw INT_MAX;
    }
    return (a/(double)b);
}

int main(){

    //************************************************
    //    ---- Throwing and Catching Exceptions -----
    //************************************************
    cout << "---- Throwing and Catching Exceptions -----" << '\n';

    try {
        cout << division(4,3, 'b') << endl;
        cout << division(4,0, 'a') << endl;
    }
    catch (char const* msg)
    {
        cerr << msg << '\n'; // cerr is the standard error stream which is used to output the errors.
    }
    catch (const int& value)
    {
        cerr << value << '\n';
    }


```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

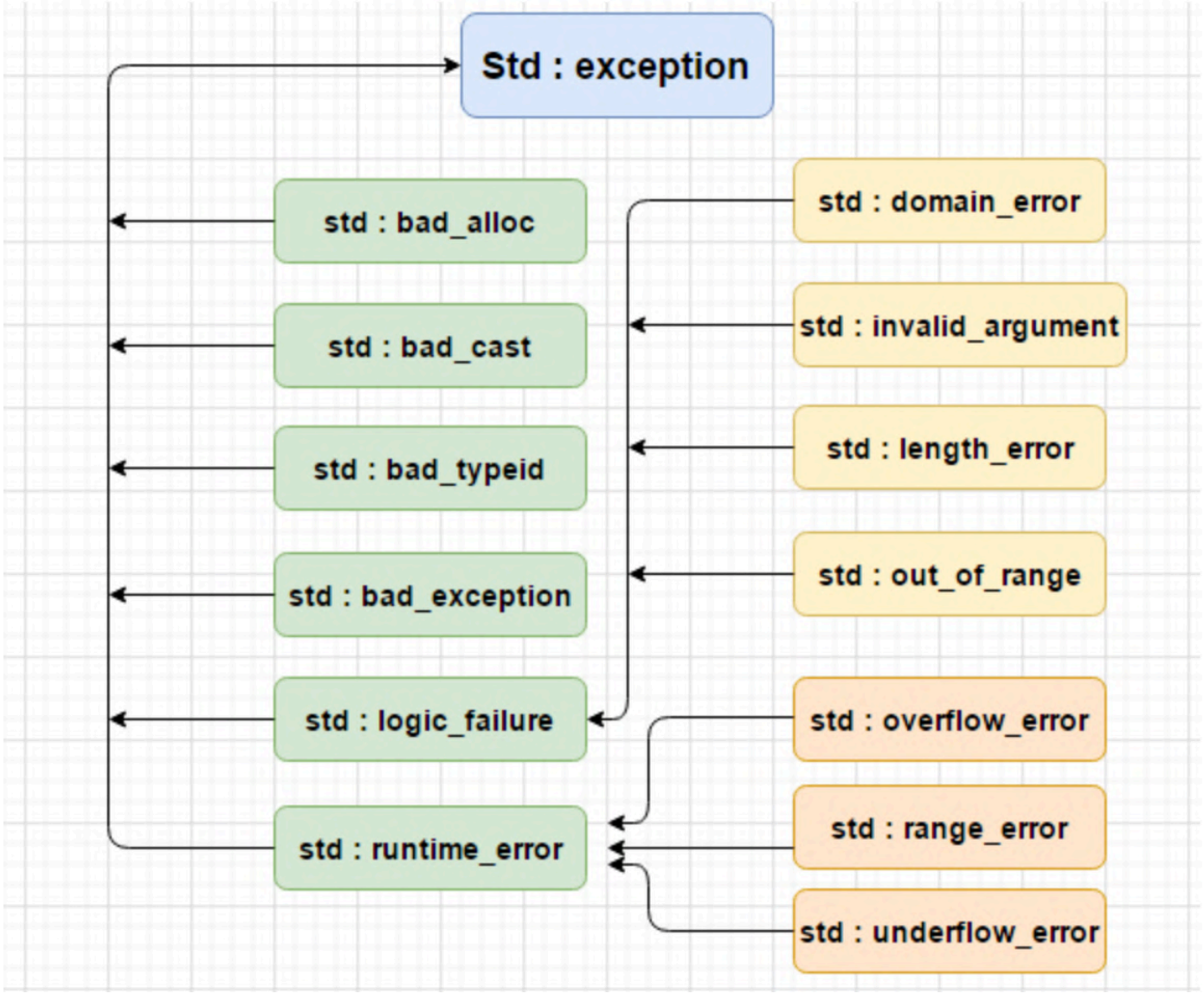
master*+ ⊗ 0 ⚠ 0 Live Share Ln 89, Col 33 Spaces: 4 UTF-8 LF C++

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

main.cpp ● | calculator.h | test.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
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
```

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

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ ⊗ 0 ⚠ 0 Live Share Ln 128, Col 1 Spaces: 4 UTF-8 LF C++

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

main.cpp ● | calculator.h | test.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
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
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ ⊗ 0 ⚠ 0 Live Share Ln 160, Col 1 Spaces: 4 UTF-8 LF C++

What

main.cpp | calculator.h | test.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
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
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
{0:1}
{1:1}
{2:1}
{3:2}
{4:3}
{5:5}
{8:21}
(base) Ivaldo:Week9 admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 136, Col 16 | Spaces: 4 | UTF-8 | LF | C++

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

main.cpp | test.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
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   }
160   catch(const std::exception &e) {
161     cerr << e.what() << '\n';
162   }
163
164
165
166
167
168
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
---- 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$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 170, Col 1 | Spaces: 4 | UTF-8 | 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

test.h > readNullPointer()

```cpp
#include "libraries.h"

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();
    }
};


void readNullPointer()
{
    int* p = nullptr;
    try
    {
        if (!p){
            throw Exc_Message("Read from nullptr");
        }
        cout << *p << endl;
    }
    catch (const std::exception& e)
    {
        cerr << e.what() << endl;
    }
}

void writeNullPointer()
{
    int* p = nullptr;
    try
    {
        if (!p){
            throw Exc_Message("Write to nullptr");
        }
        *p = 42;
    }
    catch (const std::exception& e)
```

master*+ | 0 ⚠ 0 | Live Share | Ln 25, Col 2 | Spaces: 4 | UTF-8 | LF | C++ | Mac

Test

main.cpp | test.h | libraries.h | Polygon.h | Polygon.cpp

test.h > readNullPointer()

```
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
```

master*+ | 0 ⚠ 0 | Live Share | Ln 25, Col 2 | Spaces: 4 | UTF-8 | LF | C++ | Mac

Test

main.cpp | test.h | libraries.h | Polygon.h | Polygon.cpp

main.cpp > main()

```
    cout << '{' << k << ':' << v << '}' << '\n';
}

//************************************************
//       ----  Custom exception class -----
//************************************************
cout << "---- Custom exception class -----" << '\n';

try {
    if (m.find(7) == m.end()){
        throw Exc_Message("This key is currently unavailable.");
    }

}
catch(const std::exception &e) {
  cerr << e.what() << '\n';
}

//************************************************
//         ---- More exceptions -----
//************************************************
cout << "---- More exceptions -----" << '\n';

test["RNP"](); // readNullPointer();

test["WNP"](); // writeNullPointer();

test["DWP"](); // deletedWeakPtr();

test["BFC"](); // badFunctionCall();
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

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

master*+ | 0 ⚠ 0 | Live Share | Ln 89, Col 34 (15 selected) | Spaces: 4 | UTF-8 | LF | C++ | Mac

ios::exceptions

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

ios::exceptions

main.cpp | calculator.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
182
183     // ************************************************
184     //          ---- ios::exceptions -----
185     // ************************************************
186     cout << "---- ios::exceptions -----" << '\n';
187
188     std::ifstream myfile ("test.txt");;
189     if(myfile.is_open()){
190         while (!myfile.eof()) myfile.get();
191         myfile.close();
192     }
193     else if(myfile.fail())
194     {
195         cout << "Error: Ifstream failbit or badbit is set." << endl;
196     }
197
198     std::ifstream file;
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
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

```
(base) Ivaldo:Week9 admin$ ./main
---- ios::exceptions -----
Error: Ifstream failbit or badbit is set.
```

bash
main

master*+ | 0 ⚠ 0 | Live Share | Ln 190, Col 29 (5 selected) | Spaces: 4 | UTF-8 | LF | C++

ios::exceptions

main.cpp | calculator.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

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

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
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$
```

bash
bash

master*+ ⊗ 0 ⚠ 0 Live Share Ln 210, Col 12 Spaces: 4 UTF-8 LF C++

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

main.cpp | calculator.h | Polygon.cpp | libraries.h | Polygon.h

main.cpp

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

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

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

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

bash
bash

master*+ 0 0 Live Share Ln 236, Col 1 Spaces: 4 UTF-8 LF C++

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

main.cpp | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
177
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.getNumberSides() << '\n';
196   }
197
198
199
200
201
202
203
204
205
206
207
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

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

master*+ 0 0 Live Share Ln 203, Col 1 Spaces: 4 UTF-8 LF C++

throw expression

main.cpp | Polygon.cpp | libraries.h | Polygon.h

main.cpp

```
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.getNumberSides() << '\n';
196     // }
197
198     {
199         try{
200             Polygon badPoly(-1);
201         }
202         catch(const std::exception& e) {
203             cerr << e.what() << '\n';
204         }
205     }
206
```

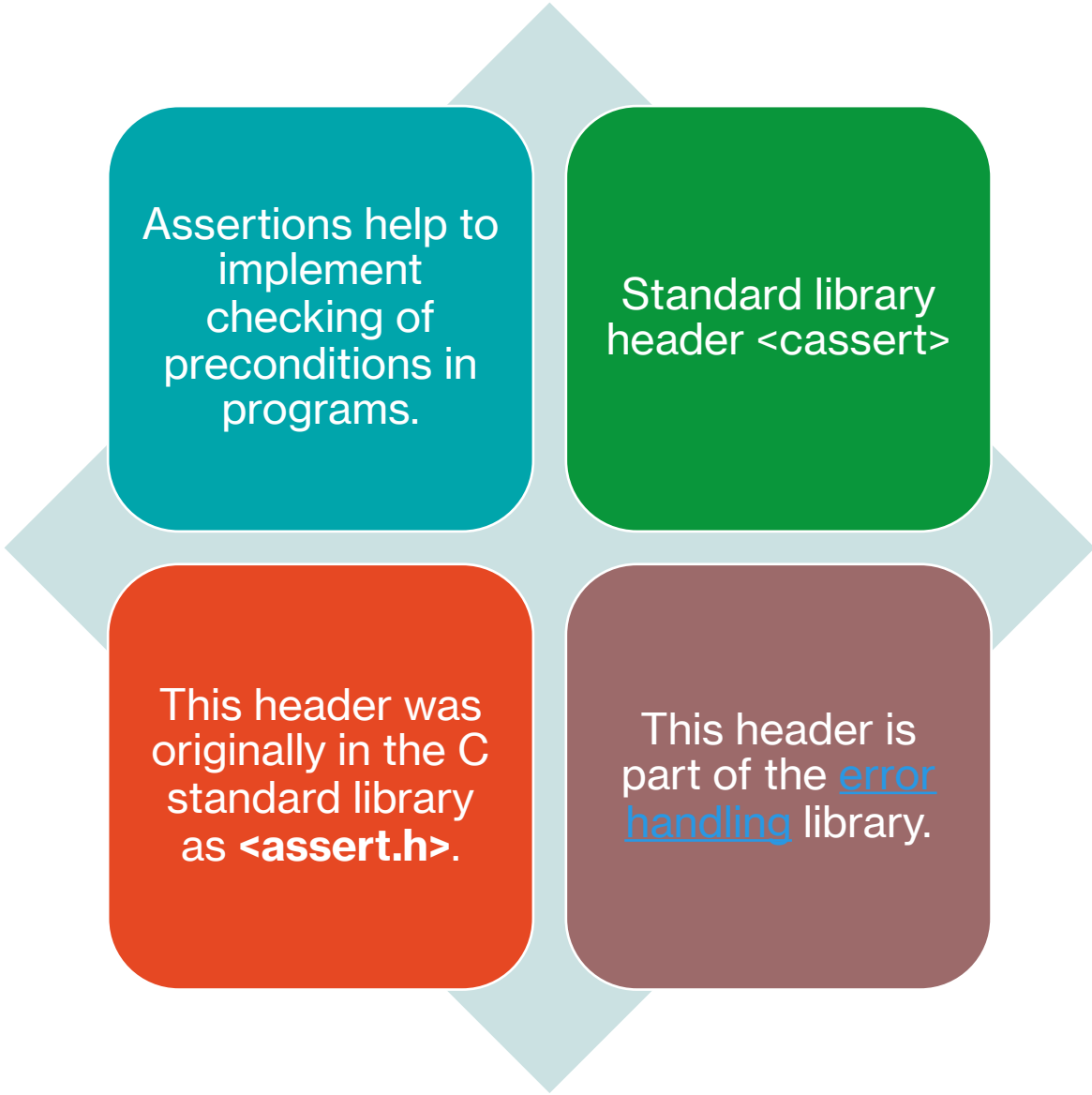
```
Polygon :: Polygon(unsigned int numberSides) : numberSides_(numberSides)
{
  // numberSides_ = (numberSides>2)? numberSides : 3;
  if ((numberSides < 3) || (numberSides > INT_MAX)){
      throw std::runtime_error("Polygon is a geometrical figure with three or more sides.");
  }
  cout << "Constructor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

}
```

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

bash

master*+ | 0 0 | Live Share | Ln 214, Col 1 | Spaces: 4 | UTF-8 | LF | C++

Assertions (abort instead of handling exception)



Assert

main.cpp | 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
```

```
21    Polygon :: Polygon(unsigned int numberSides) : numberSides_(numberSides)
22    {
23      assert(("Polygon is a geometrical figure with three or more sides.", numberSides_ >= 3));
24
25      // numberSides_ = (numberSides>2)? numberSides : 3;
26      // if ((numberSides < 3) || (numberSides > INT_MAX)){
27      //     throw std::runtime_error("Polygon is a geometrical figure with three or more sides.");
28      // }
29      cout << "Constructor Invoked"  << endl;
30
31    }
```

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$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 147, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac

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