```cpp
#include<iostream>
using namespace std;
//: C01:MyError.cpp
class MyError {
public:
    const string data;

    MyError(const string& msg) : data (msg) {}
};
void f() {
    // "Запускаем" объект исключения:
    throw MyError("something bad happened");
}
int main() {

    //Область в которой контролируется возникновение исключения
    try{

        f();
    }
    catch(const MyError& e) // ловля исключения
    {
        cout<<e.data<<endl;
    }

} ///:~



//exceptions1
#include <iostream>
using namespace std;
class DivideByZeroException
{
};
int divide(int a, int b)
{
    if (b == 0) throw DivideByZeroException();
    return a / b;
}
int main(int argc, char** argv)
{
    try
    {
        cout << divide(5, 0) << endl;
    } catch (DivideByZeroException)
    {
        cout << "error: cant divide by Zero";
    }
    return (EXIT_SUCCESS);
}

//Обработчик исключений базового класса перехватывает исключения производного класса
//exceptions2
#include <iostream>
using namespace std;
class MyException
{
};
class DivideByZeroException : public MyException
{
};
int divide(int a, int b)
{
    if (b == 0) throw DivideByZeroException();
    return a / b;
}
int main(int argc, char** argv)
{
    try
    {
        cout << divide(5, 0) << endl;
```

```
    } catch (MyException)
    {
        cout << "error";
    }
    return (EXIT_SUCCESS);
}

// exceptions3
#include <iostream>
#include <cmath>
using namespace std;
class DivideByZeroException
{
};
float divide(float a, float b)
{
    if (b == 0) throw DivideByZeroException();
    return a / b;
}
/*
 * side^2*n/(4*tan(PI/n))
 */
float RegPolyArea(float side, int n)
{
    float numerator = side * side* n;
    float denominator = 4 * tan(divide(M_PI, n));
    return divide(numerator, denominator);
}
int main(int argc, char** argv)
{
    float a;
    int b;
    try
    {
        cin >> a;
        cin >> b;
        cout << RegPolyArea(a, b) << endl;
    } catch (DivideByZeroException)
    {
        cout << "error: division by zero" << endl;
    }
    return (EXIT_SUCCESS);
}

//exceptions4
#include <stdlib.h>
#include <string>
#include <iostream>
using namespace std;
class MyException
{
    string cause;
public:
    MyException(string c)
    {
        cause = c;
    }
    string getCause()
    {
        return cause;
    }
};
class BadInput : public MyException
{
public:
    BadInput(string c) : MyException(c)
    {
    };
};
float square(float a, float b)
{
    if (a <= 0) throw MyException("a must be positive");
```

```
145      if (b <= 0) throw MyException("b must be positive");
146      //if (a < b) throw 10;
147      return a*b;
148  }
149  int main(int argc, char** argv)
150  {
151      try
152      {
153          float a, b;
154          cin >> a;
155          if (!cin.good()) throw BadInput("is not float");
156          cin >> b;
157          if (!cin.good()) throw BadInput("is not float");
158          cout << "square=" << square(a, b) << endl;
159      } catch (BadInput ex)
160      {
161          cout << "input error:" << ex.getCause() << endl;
162      } catch (MyException ex)
163      {
164          cout << "error:" << ex.getCause() << endl;
165      } catch (...)
166      {
167          cout << "undefined error" << endl;
168      }
169      return (EXIT_SUCCESS);
170  }
171
172
173
174  //Исключения могут быть выброшены повторно при помощи ключевого слова throw без агрументов, это
     может быть необходимо для управления ресурсами.
175
176  //ExceptionFiles2
177  #include <iostream>
178  #include <fstream>
179  #include <string>
180  using namespace std;
181
182  int getMaxFromFile(string filename) throw(ios_base::failure)
183  {
184      ifstream file(filename.c_str());
185      file.exceptions(ios_base::badbit|ios_base::failbit);
186      int max;
187      int i;
188      try
189      {
190          file >> i;
191          max = i;
192          while (!file.eof())
193          {
194              file >> i;
195              if (i > max) max = i;
196          }
197      } catch (...)
198      {
199          file.close();
200          throw;
201      }
202      file.close();
203      return max;
204  }
205  /*
206   *
207   */
208  int main(int argc, char** argv)
209  {
210      try
211      {
212          cout << getMaxFromFile("in.txt") << endl;
213      } catch (ios::failure& ex)
214      {
215          cout <<"error while reading file"<< endl;
```

```
216       }
217       return 0;
218   }
219
220   // Хотя на самом деле в данном примере это не нужно так как деструктор автоматически закроет
      файл. (Подробнее об этом в теме про RAII)
221
222
223   //: C01:StdExcept.cpp
224   // Создание класса исключения, производного от std::runtime_error
225   #include <stdexcept>
226   #include <iostream>
227   using namespace std;
228   class MyError : public runtime_error {
229   public:
230     MyError(const string& msg = "") : runtime_error(msg) {}
231   };
232   int main() {
233     try {
234       throw MyError("my message");
235     }
236     catch (MyError& x) {
237       cout << x.what() << endl;
238     }
239   } ///:~
240
241
242
243   //: C01:Cleanup.cpp
244   // При запуске исключения уничтожаются только готовые объекты
245   #include <iostream>
246   using namespace std;
247   class Trace {
248     static int counter;
249     int objid;
250   public:
251     Trace() {
252       objid = counter++;
253       cout << "constructing Trace #" << objid << endl;
254       if(objid == 3) throw 3;
255     }
256     ~Trace() {
257       cout << "destructing Trace #" << objid << endl;
258     }
259   };
260   int Trace::counter = 0;
261   int main() {
262     try {
263       Trace n1;
264       // Запуск исключения:
265       Trace array[5];
266       Trace n2;  // Сюда не попадаем
267     } catch(int i) {
268       cout << "caught " << i << endl;
269     }
270   } ///:~
271
272   // constructing Trace #0
273   // constructing Trace #1
274   // constructing Trace #2
275   // constructing Trace #3
276   // destructing Trace #2
277   // destructing Trace #1
278   // destructing Trace #0
279   // caught 3
280
281
282
```