Задание(для удобства задание разбито на пять частей).

Во всех заданиях реализуемые функции должны генерировать подходящие исключения. Обработку исключений нужно выполнять главной функцией, которая должна демонстрировать обработку всех перехватываемых исключений.

Функции, реализуемые в заданиях, обязаны выполнять проверку передаваемых параметров и генерировать исключения в случае ошибочных. Все функции реализуются в четырех вариантах:

- без спецификации исключений;
- со спецификацией throw();
- с конкретной спецификацией с подходящим стандартным исключением;
- спецификация с собственным реализованным исключением.

Собственное исключение должно быть реализовано в трех вариантах: как пустой класс, как независимый класс с полями-параметрами функции, как наследник от стандартного исключения с полями.

15. Функция вычисляет углы прямоугольного треугольника. В качестве параметров передаются катеты A и B. (Синус угла A1, противолежащего катету A, вычисляем по формуле sin(A1)=a/c, где c - гипотенуза треугольника.)

Часть 1(без спецификации throw()).

ExceptionFunc.h

```h
#include <iostream>
#include <cmath>

struct MyTriangle
{
   double a, b, c;
   double angle_a, angle_b;

   MyTriangle(double ua, double ub)
   {
        a = ua;
        b = ub;
   }
};
```

ExceptionFunc.cpp

```cpp
#include "ExceptionFunc.h"

using namespace std;

void Angles(MyTriangle triangle)
{
```

```
    if (isnan(triangle.a)) throw 1.0;
    if (isnan(triangle.b)) throw 1.0;
    triangle.c = sqrt(pow(triangle.a, 2) * pow(triangle.b, 2));
    triangle.angle_a = triangle.a / triangle.c;
    triangle.angle_b = triangle.b / triangle.c;
    cout << triangle.angle_a << " " << triangle.angle_b;
}

int main()
{
    double a = NAN, b = NAN;
    MyTriangle test(a, b);
    try
    {
        Angles(test);
    }
    catch (double)
    {
        cout << "Error!\n";
    }
    return 0;
}
```

Результат.

```
Error!
```

Часть 2(с спецификацией throw()).

ExceptionFunc.h

```
#include <iostream>
#include <cmath>

struct MyTriangle
{
    double a, b, c;
    double angle_a, angle_b;

    MyTriangle(double ua, double ub)
    {
```

```
        a = ua;
        b = ub;
    }
};
```

ExceptionFunc.cpp

```
#include "ExceptionFunc.h"

using namespace std;

void Angles(MyTriangle triangle)
{
    if (isnan(triangle.a)) throw invalid_argument("a is NaN");
    if (isnan(triangle.b)) throw invalid_argument("b is NaN");
    triangle.c = sqrt(pow(triangle.a, 2) * pow(triangle.b, 2));
    triangle.angle_a = triangle.a / triangle.c;
    triangle.angle_b = triangle.b / triangle.c;
    cout << triangle.angle_a << " " << triangle.angle_b;
}

int main()
{
    double a = NAN, b = NAN;
    MyTriangle test(a, b);
    try
    {
        Angles(test);
    }
    catch (invalid_argument e)
    {
        cout << "Error!\n" << e.what() << endl;
    }
    return 0;
}
```

Результат.

```
Error!
a is NaN
```

Часть 3(Специфицируем исключение собственным реализованным исключением (пустой класс)).

ExceptionFunc.h

```
#include <iostream>
#include <cmath>

struct MyTriangle
{
   double a, b, c;
   double angle_a, angle_b;

   MyTriangle(double ua, double ub)
   {
       a = ua;
       b = ub;
   }
};

class MyEmptyException{};
```

ExceptionFunc.cpp

```
#include "ExceptionFunc.h"

using namespace std;

void Angles(MyTriangle triangle)
{
   if (isnan(triangle.a)) throw MyEmptyException();
   if (isnan(triangle.b)) throw MyEmptyException();
   triangle.c = sqrt(pow(triangle.a, 2) * pow(triangle.b, 2));
   triangle.angle_a = triangle.a / triangle.c;
   triangle.angle_b = triangle.b / triangle.c;
   cout << triangle.angle_a << " " << triangle.angle_b;
}

int main()
{
   double a = NAN, b = NAN;
   MyTriangle test(a, b);
   try
   {
       Angles(test);
```

```
    }
    catch (MyEmptyException e)
    {
        cout << "Error!\n";
    }
    return 0;
}
```

Результат.

```
Error!
```

Часть 4(Специфицируем исключение собственным реализованным исключением (класс с полями)).

ExceptionFunc.h

```
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

struct MyTriangle
{
    double a, b, c;
    double angle_a, angle_b;

    MyTriangle(double ua, double ub)
    {
        a = ua;
        b = ub;
    }
};

class MyTriangleException
{
    double a, b;

    public:
```

```
        string message;
        MyTriangleException(double ua, double ub, const string& s)
        {
            a = ua;
            b = ub;
            message = s;
        }
};
```

ExceptionFunc.cpp

```
#include "ExceptionFunc.h"

using namespace std;

void Angles(MyTriangle triangle)
{
   if (isnan(triangle.a)) throw MyTriangleException(triangle.a,
triangle.b, "a is NAN");
   if (isnan(triangle.b)) throw MyTriangleException(triangle.a,
triangle.b, "b is NAN");
   triangle.c = sqrt(pow(triangle.a, 2) * pow(triangle.b, 2));
   triangle.angle_a = triangle.a / triangle.c;
   triangle.angle_b = triangle.b / triangle.c;
   cout << triangle.angle_a << " " << triangle.angle_b;
}

int main()
{
   double a = NAN, b = NAN;
   MyTriangle test(a, b);
   try
   {
       Angles(test);
   }
   catch (MyTriangleException e)
   {
       cout << e.message << endl;
   }
   return 0;
}
```

Результат.

```
a is NAN
```

Часть 5(Специфицируем исключение собственным реализованным исключением (класс наследник стандартного класса-исключения)).

ExceptionFunc.h

```
#include <iostream>
#include <cmath>
#include <string>
#include <exception>

using namespace std;

struct MyTriangle
{
   double a, b, c;
   double angle_a, angle_b;

   MyTriangle(double ua, double ub)
   {
       a = ua;
       b = ub;
   }
};

class MyTriangleException : public invalid_argument
{
   double a, b;

   public:
       string message;
       MyTriangleException(double ua, double ub, const string& s) :
a(ua), b(ub), invalid_argument(s) {};
};
```

ExceptionFunc.cpp

```
#include "ExceptionFunc.h"

using namespace std;

void Angles(MyTriangle triangle)
{
   if (isnan(triangle.a)) throw MyTriangleException(triangle.a,
triangle.b, "a is NAN");
   if (isnan(triangle.b)) throw MyTriangleException(triangle.a,
triangle.b, "b is NAN");
   triangle.c = sqrt(pow(triangle.a, 2) * pow(triangle.b, 2));
   triangle.angle_a = triangle.a / triangle.c;
   triangle.angle_b = triangle.b / triangle.c;
   cout << triangle.angle_a << " " << triangle.angle_b;
}

int main()
{
   double a = NAN, b = NAN;
   MyTriangle test(a, b);
   try
   {
       Angles(test);
   }
   catch (MyTriangleException e)
   {
       cout << e.what() << endl;
   }
   return 0;
}
```

Результат.

```
a is NAN
```