МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ
Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

Лабораторная работа № 2
по курсу «Объектно-ориентированное программирование»
«Классы и объекты в С++»
8 ВАРИАНТ

Выполнили:
студенты гр. КТбо2-1
Яшенков А.В.

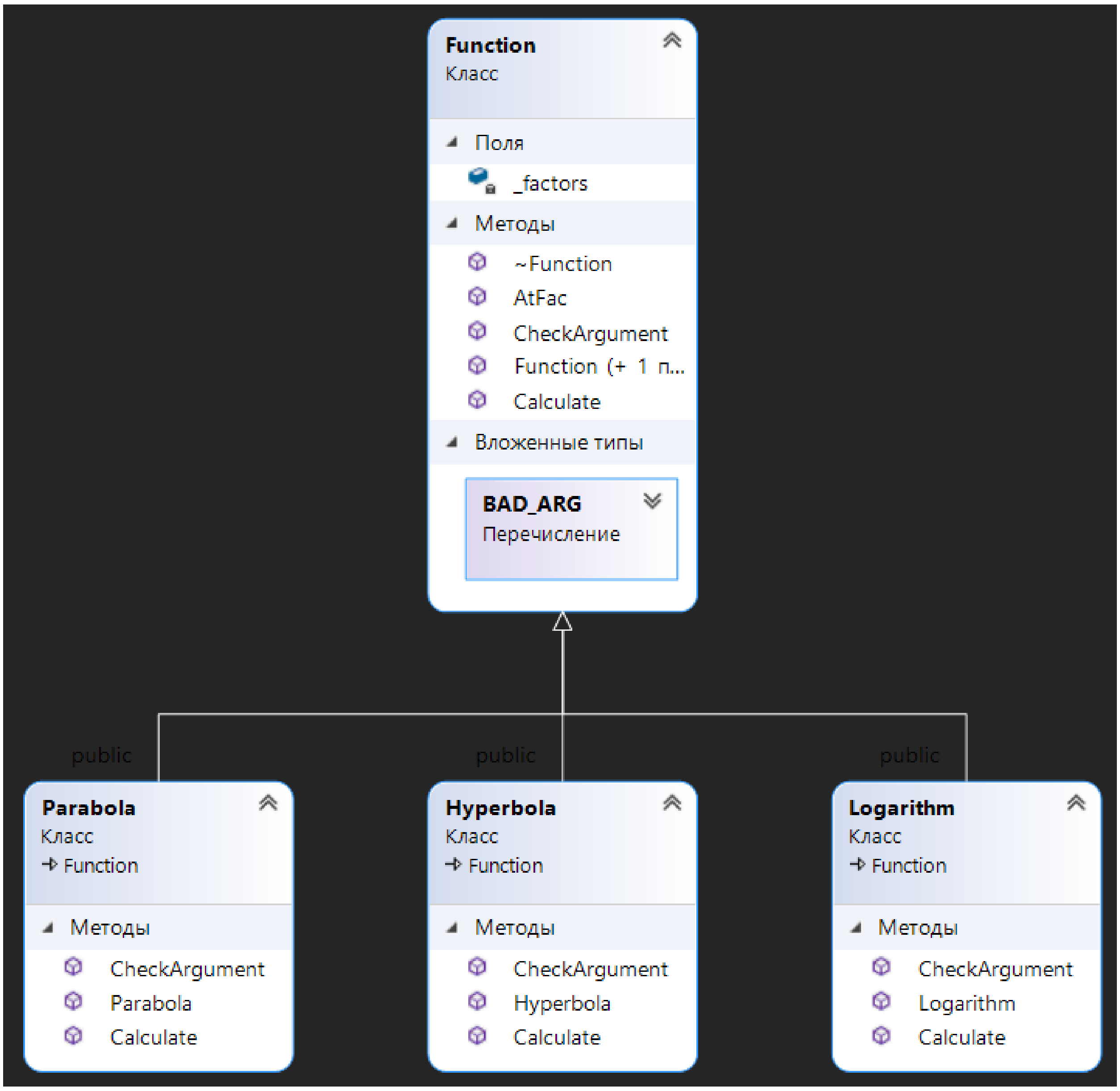
Проверил:
Тарасов С. А.

Таганрог 2020

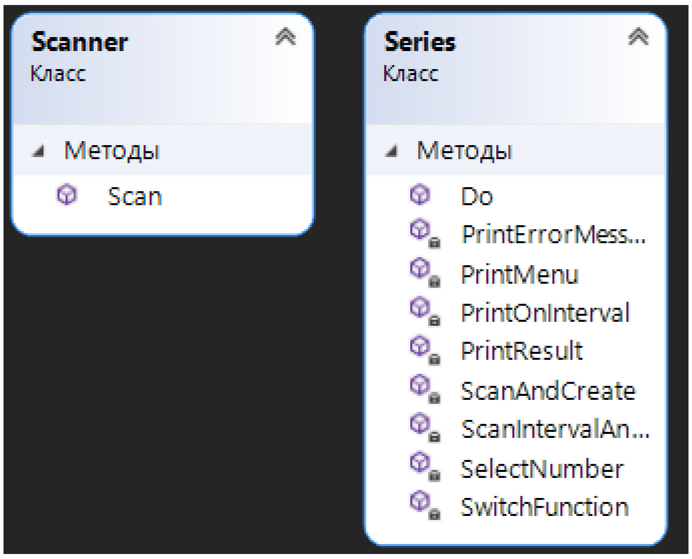
1. Вариант задания №8

Создать класс Function(функция) с методами вычисле-ния значения функции y= f(x) в заданной точке x и вывода ре-зультата на консоль. На его основе определить классы Ellipse, Hiperbola и Parabola, в которых реализуются соответствующие математические зависимости. В дополнительном классе Seriesвсе три функции должны вызываться для заданного интервала изменения x с выводом результатов.

2. UML-диаграмма наследования классов Parabola, Hiperbola, Logarithm от класса Function.



3. UML-диаграммы классов Series, Scanner.



4. Листинг

main.cpp:

```
#include "Series.h"

int main()

{

    Series series;

    series.Do();



    return 0;

}

#pragma once

#include "Logarithm.h"

#include "Hyperbola.h"

#include "Parabola.h"
```

Series.h:

```
class Series
{
public:
	void Do() const;


private:
	Function* ScanAndCreate(char function) const;
	void SelectNumber(int& number) const;
	void PrintMenu() const;
	void SwitchFunction(Function** function, int number) const;
	void ScanIntervalAndStep(double& start, double& end, double& step) const;
	void PrintOnInterval(double& start, double& end, double& step, Function* function) const;
	void PrintResult(double x, double result, Function* function) const;
	void PrintErrorMessage(double x, const char* message, Function* function) const;
};
```

Series.cpp:

```
#include "Series.h"
#include "Scanner.h"
#include <typeinfo>
#include <iostream>


void Series::Do() const
```

```
{
	Function* FunctionPointer = nullptr;

	int number = INT_MAX;

	while (number)
	{

		SelectNumber(number);

		if (!number)
		{
			break;
		}

		SwitchFunction(&FunctionPointer, number);

		try
		{
			FunctionPointer->CheckArgument();
		}
		catch (const std::exception& err)
		{
			std::cerr << err.what() << std::endl;
```

```
                system("pause");

                system("cls");

                continue;

            }


            double start, end;

            double step = -1;


            ScanIntervalAndStep(start, end, step);


            PrintOnInterval(start, end, step, FunctionPointer);


            std::cout << std::endl;


            system("pause");

            system("cls");

            delete FunctionPointer;

            FunctionPointer = nullptr;

        }

}


Function* Series::ScanAndCreate(char function) const

{

        if (function == 'P')
```

```
	{
		double factors[3];


		Scanner().Scan("%f%f%f", &factors[0], &factors[1],
&factors[2]);


		return new Parabola(factors[0], factors[1], factors[2]);
	}


	if (function == 'L')
	{
		double factor;


		Scanner().Scan("%f", &factor);


		return new Logarithm(factor);
	}


	if (function == 'H')
	{
		double factor;


		Scanner().Scan("%f", &factor);


		return new Hyperbola(factor);
	}

	throw std::logic_error("Uncorrect argument!");
}

void Series::SelectNumber(int& number) const
{
	bool condition = true;

	while (condition)
	{
		try
		{
			PrintMenu();

			Scanner().Scan("%d", &number);

			condition = number != 0 && number != 1 && number != 2
&& number != 3;

			if (condition)
			{
				std::cout << "Repeat please!\n\n";
			}
		}
		catch (const std::exception& err)

		{

			std::cerr << err.what() << std::endl;

			std::cout << "Repeat please!\n\n";

		}

	}

}


void Series::PrintMenu() const

{

	std::cout << "1. Hyperbola\n";

	std::cout << "2. Logarithm\n";

	std::cout << "3. Parabola\n";

	std::cout << "0. Exit\n";

	std::cout << "Select function: ";

}


void Series::SwitchFunction(Function** function, int number) const

{

	switch (number)

	{

	case 1:

	{

		bool scanFlag = false;
```

```
        std::cout << "\nEnter a factor of Hyperbola (One number): ";

        while (!scanFlag)
        {
                try
                {
                        *function = ScanAndCreate('H');

                        scanFlag = true;
                        std::cout << std::endl;
                }
                catch (const std::exception& err)
                {
                        std::cerr << err.what() << std::endl;
                        std::cout << "Repeat please!\n";
                        std::cout << "Your factor: ";
                }
        }
}
break;
case 2:
{
        bool scanFlag = false;
        std::cout << "\nEnter a base of the Logarithm (One number): ";

        while (!scanFlag)
        {
            try
            {

                *function = ScanAndCreate('L');

                scanFlag = true;
                std::cout << std::endl;
            }
            catch (const std::exception& err)
            {
                std::cerr << err.what() << std::endl;
                std::cout << "Repeat please!\n";
                std::cout << "Your factor: ";
            }
        }
    }
    break;
    case 3:
    {
```

```
                bool scanFlag = false;


                std::cout << "\nEnter a factor of Parabola(Three numbers
separated by a space. Example: 1 2 3): ";


                while (!scanFlag)
                {
                        try
                        {
                                *function = ScanAndCreate('P');


                                scanFlag = true;
                                std::cout << std::endl;
                        }
                        catch (const std::exception& err)
                        {
                                std::cerr << err.what() << std::endl;
                                std::cout << "Repeat please!\n";
                                std::cout << "Your factor: ";
                        }
                }
        }
        break;
        default:
                break;
        }
}


void Series::ScanIntervalAndStep(double& start, double& end, double&
step) const
{
        std::cout << "Enter an interval value.\n";


        while (true) {
                try
                {


                        std::cout << "Start: ";
                        Scanner().Scan("%f", &start);


                        std::cout << "End: ";
                        Scanner().Scan("%f", &end);


                        std::cout << "Enter a step: ";
                        Scanner().Scan("%f", &step);


                        if (step > 0 && start <= end)
                        {
                                break;
                        }
```

```
				else
				{
					std::cout << "Uncorrect value!\n";
					std::cout << "Repeat please!\n\n";
				}
			}
			catch (const std::exception& err)
			{
				std::cerr << err.what() << std::endl;

				std::cout << "Repeat please!\n\n";
			}
		}

}

void Series::PrintOnInterval(double& start, double& end, double& step,
Function* function) const
{
	std::cout << std::endl;
	std::string name = typeid(*function).name();
	name.erase(0, 6);

	for (double x = start; x <= end; x += step)
	{
		try
		{
			/*double result = function->Calculate(x);
			std::cout << name << "(" << x << ") = " << result << 
";\n";*/

			PrintResult(x, function->Calculate(x), function);
		}
		catch (const std::exception& err)
		{
			//std::cerr << name << "(" << x << ") = " << err.what() << 
std::endl;
			PrintErrorMessage(x, err.what(), function);
		}
	}
}

void Series::PrintResult(double x, double result, Function* function) const
{
	std::string name = typeid(*function).name();
	name.erase(0, 6);



	if (name == "Hyperbola")
	{
```

```
                std::cout << name << "( " << function->AtFac(0) << "/x ) = " <<
result << ", where x = " << x << std::endl;

        }



        if (name == "Logarithm")

        {

                std::cout << name << "( Log" << function->AtFac(0) << "(x) ) = "
<< result << ", where x = " << x << std::endl;

        }



        if (name == "Parabola")

        {

                std::cout << name << "( " << function->AtFac(0) <<"*x^2 + " <<
function->AtFac(1) <<"*x + " << function->AtFac(2) << " ) = " << result <<
", where x = " << x << std::endl;

        }

}



void Series::PrintErrorMessage(double x, const char* message, Function*
function) const

{

        std::string name = typeid(*function).name();

        name.erase(0, 6);



        if (name == "Hyperbola")

        {
            std::cout << name << "( " << function->AtFac(0) << "/x ) = " <<
message << ", where x = " << x << std::endl;

      }



      if (name == "Logarithm")

      {

            std::cout << name << "( Log" << function->AtFac(0) << "(x) ) = "
<< message << ", where x = " << x << std::endl;

      }

}
```

Funtion.h:

```
#pragma once

#include <stdexcept>

#include <cmath>



class Function

{

public:

      Function() = default;

      Function(int size);



      virtual double Calculate(double x) const = 0;

      virtual void CheckArgument() const = 0;

      virtual ~Function();

      double& AtFac(int index) const;
```

```
    enum BAD_ARG
    {
        ZERO,
        ONE
    };

private:
    double* _factors = nullptr;
};
```

Function.cpp:

```
#include "Function.h"

Function::Function(int size)
    : _factors(new double[size]) {}

Function::~Function()
{
    if (_factors)
    {
        delete[] _factors;
    }
}
```

```
double& Function::AtFac(int index) const
{
	return _factors[index];
}
Hyperbola.h:
#pragma once
#include "Function.h"

class Hyperbola : public Function
{
public:
	Hyperbola(double a = 0.);

	void CheckArgument() const;
	double Calculate(double x) const;
};
Hyperbola.cpp:
#include "Hyperbola.h"
#define HYP_FAC_ARG 1
#define HYP_FAC 0

Hyperbola::Hyperbola(double a)
	: Function(HYP_FAC_ARG)
{
	AtFac(HYP_FAC) = a;
}

double Hyperbola::Calculate(double x) const
{
	if (x == BAD_ARG::ZERO)
	{
		throw std::invalid_argument("Division by zero!");
	}
	else
	{
		return AtFac(HYP_FAC) / x;
	}
}

void Hyperbola::CheckArgument() const {}
```

Logarithm.h:

```
#pragma once
#include "Function.h"

class Logarithm : public Function
{
public:
	Logarithm(double a = 2.);
	void CheckArgument() const;

	double Calculate(double x) const;

};

Logarithm.cpp:

#include "Logarithm.h"

#define BASE_ARG 1

#define BASE 0


Logarithm::Logarithm(double a)

	: Function(BASE_ARG)

{

	AtFac(BASE) = a;

}


double Logarithm::Calculate(double x) const

{

	if (x <= BAD_ARG::ZERO)

	{

		throw std::invalid_argument("The parameter is less than or equal to zero");

	}



	return std::log(x) / std::log(AtFac(BASE));

}
```

```
void Logarithm::CheckArgument() const
{
        if (AtFac(BASE) == BAD_ARG::ONE)
        {
                throw std::invalid_argument("The base is equal to one!");
        }

        if (AtFac(BASE) <= BAD_ARG::ZERO)
        {
                throw std::invalid_argument("Base less than or equal to zero");
        }
}
```

Parabola.h:

```
#pragma once
#include "Function.h"

class Parabola : public Function
{
public:
        Parabola(double a = 0., double b = 0., double c = 0.);

        void CheckArgument() const;
        double Calculate(double x) const;
};
```

Parabola.cpp:

```
#include "Parabola.h"
#define PAR_FAC 3

Parabola::Parabola(double a, double b, double c)
	: Function(PAR_FAC)
{
	AtFac(0) = a;
	AtFac(1) = b;
	AtFac(2) = c;
}

double Parabola::Calculate(double x) const noexcept
{
	 return AtFac(0) * pow(x, 2) + AtFac(1) * x + AtFac(2);
}

void Parabola::CheckArgument() const {}
```

Scanner.h:

```
#pragma once

class Scanner
{
```

```
public:
	int Scan(const char*, ...);
};
Scanner.cpp:
#include "Scanner.h"
#include <iostream>
#include <stdarg.h>
#include <cstring>
#include <string>
#include <sstream>

int Scanner::Scan(const char* format, ...)
{
	std::string input;
	std::getline(std::cin, input);

	std::istringstream sin(input);

	va_list arg_pointer;
	va_start(arg_pointer, format);

	int count_per_cent = 0;

	for (int i = 0; i < strlen(format); ++i)
```

```
{
        if (format[i] == '%')
        {
                count_per_cent++;
                i++;
        }
}

if (!count_per_cent)
{
        return count_per_cent;
}

for (int i = 0; i < strlen(format) - 1; i++)
{
        if (format[i] == '%')
        {
                i++;

                switch (format[i])
                {
                case 'd':
                        {
                                if (!(sin >> *(int*)va_arg(arg_pointer, int*)))
```

```
                    {
                            throw std::logic_error("Uncorrect
value!");
                    }
                    count_per_cent++;
            }
            break;
        case 'c':
            {
                    if (!(sin >> *(char*)va_arg(arg_pointer, char*)))
                    {
                            throw std::logic_error("Uncorrect
value!");
                    }
                    count_per_cent++;
            }
            break;
        case 's':
            {
                    if (!(sin >> *(std::string*)va_arg(arg_pointer,
std::string*)))
                    {
                            throw std::logic_error("Uncorrect
value!");
                    }
                    count_per_cent++;
```

```
                }
                break;
            case 'f':
                {
                    if (!(sin >> *(double*)va_arg(arg_pointer,
double*)))
                    {
                        throw std::logic_error("Uncorrect
value!");
                    }
                    count_per_cent++;
                }
                break;
            default:
                throw std::logic_error("Uncorrect line!");
                break;
            }
        }
        else if (format[i] == ' ')
        {
            continue;
        }
        else
        {
            throw std::logic_error("Uncrorrect format!");
        }

    }

    std::string test_end;

    if (sin >> test_end)
    {
        throw std::logic_error("Uncorrect line!");
    }

    va_end(arg_pointer);

    return count_per_cent;
}
```