

Колледж космического машиностроения и технологий

**ОТЧЕТ**

по учебной практике

УП.03.01. «Участие в интеграции программных модулях»

по специальности 09.02.03 «Программирование в компьютерных системах»

Выполнил студент

Волков М.А.

Группа П1-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Принял преподаватель

Гусятинер Л.Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Подпись)

**Королев 2021 г.**

**Содержание**

[Раздел 1. Проектирование и разработка программы «Калькулятор» 2](#_Toc91514194)

[1.1 Коллективная разработка программы 2](#_Toc91514195)

[1.2 Построение UML-диаграмм к программе «Калькулятор» 3](#_Toc91514196)

[1.3 Разработка модуля «math\_ast.py» 4](#_Toc91514197)

[Раздел 2. Проектирование и разработка программы «Блокнот». 21](#_Toc91514198)

[2.1. Построение UML-диаграмм к программе «Блокнот» 21](#_Toc91514199)

[2.2. Разработка программы «Блокнот» 21](#_Toc91514200)

[Дерево файлов 27](#_Toc91514201)

[Заключение. 28](#_Toc91514202)

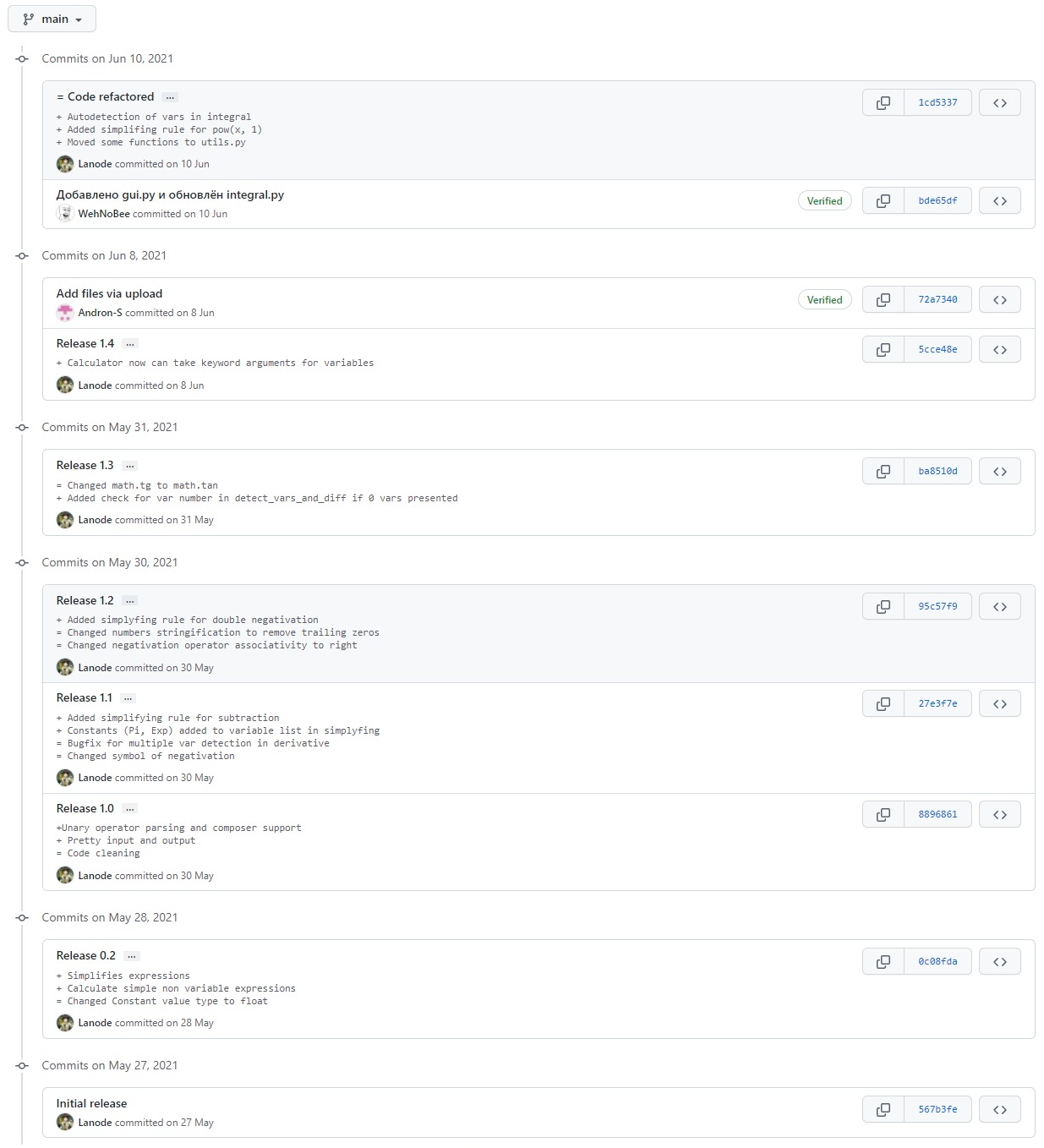
# Раздел 1. Проектирование и разработка программы «Калькулятор»

## Коллективная разработка программы

Работа над программой «Калькулятор» велась совместно четырьмя разработчиками, используя систему контроля версий Git. Разработчики параллельно разрабатывали отдельные функции программы.

Рисунок 1.

Список изменений разработчиков.



## Построение UML-диаграмм к программе «Калькулятор»

Рисунок 2.

Диаграмма классов.

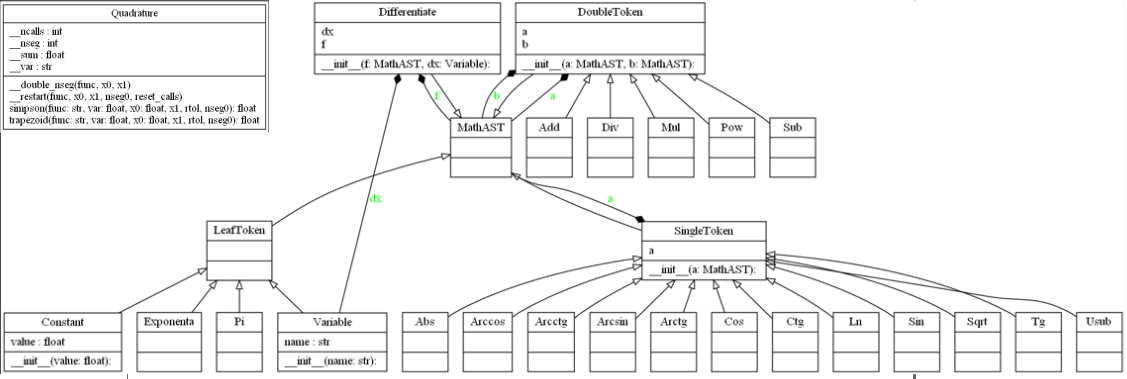
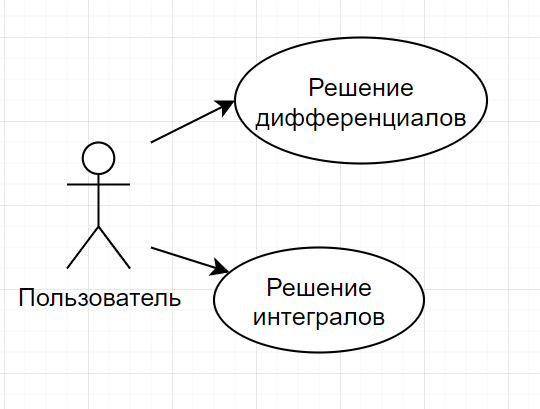


Рисунок 3.

Диаграмма прецедентов.



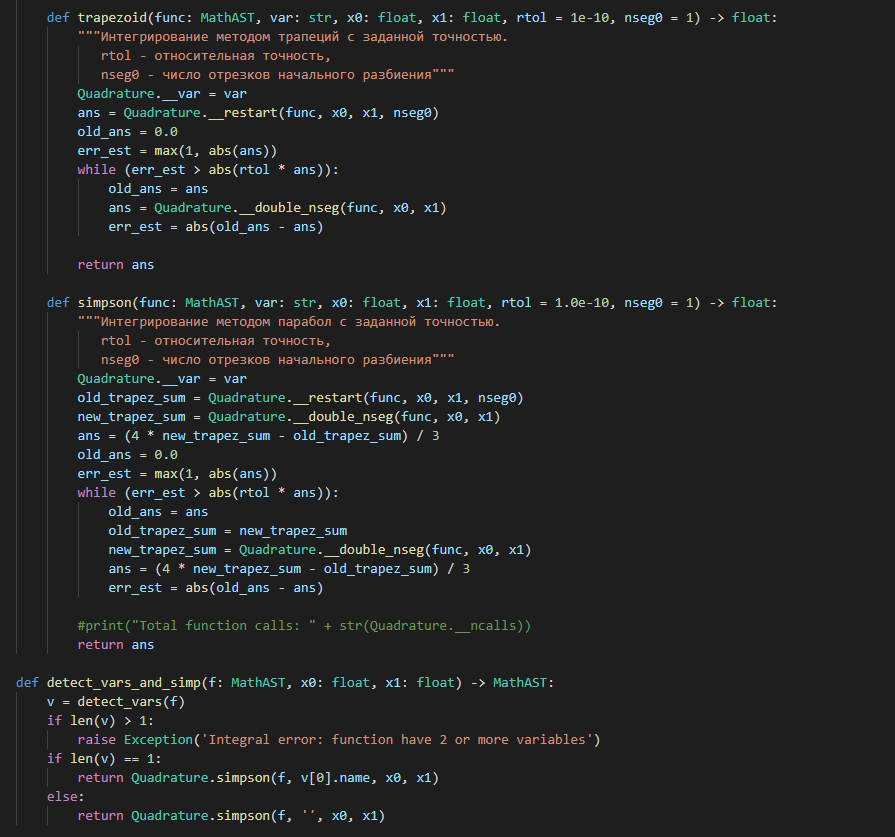
## Разработка модуля «integral.py»

Для представления математический выражений в программе был разработан модуль, решающий интегралы по формуле Симпсона.

Формула Симпсона (также Ньютона-Симпсона[1]) относится к приёмам численного интегрирования. Получила название в честь британского математика Томаса Симпсона (1710—1761). Суть метода заключается в приближении подынтегральной функции на отрезке [a, b]} [a, b] интерполяционным многочленом второй степени p\_{2} (x)} p\_{2} (x), то есть приближение графика функции на отрезке параболой. Метод Симпсона имеет порядок погрешности 4 и алгебраический порядок точности 3.

Рисунок 4.

Демонстрация функции Симпсона.



Листинг 1. main.py

from calculator import calculate

from simplifier import simplify

from derivative import differentiate, detect\_vars\_and\_diff

from math\_ast import \*

from math\_parser import parse

from math\_composer import compose

f = input('function:\n> ')

try:

ast = parse(f)

ast = simplify(ast)

print('simplified:\n'+compose(ast))

deriv\_ast = detect\_vars\_and\_diff(ast)

print('derivative:\n'+compose(deriv\_ast))

deriv\_ast = simplify(deriv\_ast)

print('simplified derivative:\n'+compose(deriv\_ast))

except Exception as e:

print(e)

Листинг 2. math\_ast.py

from dataclasses import dataclass

import typing

class MathAST: pass

class LeafToken (MathAST): pass

@dataclass

class SingleToken (MathAST):

a: MathAST

@dataclass

class DoubleToken (MathAST):

a: MathAST

b: MathAST

class Pi (LeafToken): pass

class Exponenta (LeafToken): pass

class Sin (SingleToken): pass

class Cos (SingleToken): pass

class Tg (SingleToken): pass

class Ctg (SingleToken): pass

class Ln (SingleToken): pass

class Abs (SingleToken): pass

class Usub (SingleToken): pass

class Arcsin (SingleToken): pass

class Arccos (SingleToken): pass

class Arctg (SingleToken): pass

class Arcctg (SingleToken): pass

class Sqrt (SingleToken): pass

class Mul (DoubleToken): pass

class Add (DoubleToken): pass

class Sub (DoubleToken): pass

class Div (DoubleToken): pass

class Pow (DoubleToken): pass

@dataclass

class Variable (LeafToken):

name: str

@dataclass

class Constant (LeafToken):

value: float

@dataclass

class Differentiate (MathAST):

f: MathAST

dx: Variable

constants: typing.List[typing.Type[LeafToken]] = [Pi, Exponenta]

functions: typing.List[typing.Type[SingleToken]] = [Sin, Cos, Tg, Ln, Abs, Usub, Arcsin, Arccos, Arctg, Arcctg]

Листинг 3. utils.py

from math\_ast import \*

import typing as t

def detect\_vars(f: MathAST) -> t.List[Variable]:

vars: t.List[Variable] = list()

def walk(f):

if isinstance(f, Variable):

for v in vars:

if v.name == f.name:

break

else:

vars.append(f)

elif isinstance(f, SingleToken):

walk(f.a)

elif isinstance(f, DoubleToken):

walk(f.a)

walk(f.b)

walk(f)

return vars

def is\_depends\_on\_dx(tree: MathAST, dx: str) -> bool:

if isinstance(tree, DoubleToken):

if isinstance(tree.a, Variable):

if tree.a.name == dx:

a = True

else:

a = False

else:

a = is\_depends\_on\_dx(tree.a, dx)

if isinstance(tree.b, Variable):

if tree.b.name == dx:

b = True

else:

b = False

else:

b = is\_depends\_on\_dx(tree.b, dx)

return a or b

# (e.a match {

# case Variable(name) => if(name == dx) true else false

# case \_ => isDependsOnVar(e.a)

# })||(e.b match {

# case Variable(name) => if(name == dx) true else false

# case \_ => isDependsOnVar(e.b)

# })

elif isinstance(tree, SingleToken): return is\_depends\_on\_dx(tree.a, dx)

elif isinstance(tree, Variable):

if tree.name == dx:

return True

else:

return False

else:

return False

def is\_var\_dependant(tree: MathAST) -> bool:

if isinstance(tree, DoubleToken):

a = is\_var\_dependant(tree.a)

b = is\_var\_dependant(tree.b)

return a or b

elif isinstance(tree, SingleToken):

return is\_var\_dependant(tree.a)

elif isinstance(tree, Variable) or isinstance(tree, (Pi, Exponenta)):

return True

else:

return False

def is\_leaf(e: MathAST) -> bool:

if isinstance(e, Variable) or isinstance(e, Constant):

return True

elif isinstance(e, Pi) or isinstance(e, Exponenta):

return True

else:

return False

Листинг 4. math\_parser.py

from math\_ast import \*

import typing as t

from enum import IntEnum, auto

import re

class TokenTypes(IntEnum):

NAMED\_FUNCTION = auto()

CONSTANT = auto()

OPERATOR = auto()

UNARY\_OPERATOR = auto()

LEFT\_PARENTHESIS = auto()

RIGHT\_PARENTHESIS = auto()

NUMBER = auto()

VARIABLE = auto()

rules = [

{

'key': r"[\~]",

'type': TokenTypes.OPERATOR,

'data': {

'args': 1,

'precedence': 3,

'isLeftAssociative': False

}

},

{

'key': r"sin|cos|tg|ctg|log|sqrt",

'type': TokenTypes.NAMED\_FUNCTION,

'data': {

'args': 1,

'precedence': 4,

'isLeftAssociative': True

}

},

{

'key': r"pi|e",

'type': TokenTypes.CONSTANT

},

{

'key': r"[\^]",

'type': TokenTypes.OPERATOR,

'data': {

'args': 2,

'precedence': 3,

'isLeftAssociative': True

}

},

{

'key': r"[\*\/]",

'type': TokenTypes.OPERATOR,

'data': {

'args': 2,

'precedence': 2,

'isLeftAssociative': True

}

},

{

'key': r"[+-]",

'type': TokenTypes.OPERATOR,

'data': {

'args': 2,

'precedence': 1,

'isLeftAssociative': True

}

},

{ 'key': r"[([]", 'type': TokenTypes.LEFT\_PARENTHESIS },

{ 'key': r"[)\]]", 'type': TokenTypes.RIGHT\_PARENTHESIS },

{ 'key': r"[0-9.,]+", 'type': TokenTypes.NUMBER },

{ 'key': r"[a-zA-Z]", 'type': TokenTypes.VARIABLE }

]

def print\_tokens(t):

for x in t:

print(x[0], end=' ')

print()

# вохзможно надо переделать в более продвинутый отокенайзер не на регексах

# а на лямбдах с передачей очереди уже чсщуествующих токенов в нее

def tokenize(s: str) -> typing.List[typing.Tuple[str, typing.Dict]]:

s = re.sub('(^|[\(\+\-\\*/\^])\-', '\g<1>~', s)

s = re.sub('(^|[\(\+\-\\*/\^])\+', '\g<1>', s)

output: typing.List[typing.Tuple[str, typing.Dict]] = list()

start = 0

while start < len(s):

for rule in rules:

m = re.match(rule['key'], s[start:])

if m is not None:

output.append((m.group(0), rule))

start = start+m.end()

break

else:

raise Exception("Tokenization error near {}: {}".format(start, s[start:]))

#print\_tokens(output)

return output

def shunting\_yard(tokens: t.List[t.Tuple[str, t.Dict]]) -> t.List[t.Tuple[str, t.Dict]]:

op\_stack = list()

out = list()

for token in tokens:

if token[1]['type'] in (TokenTypes.CONSTANT, TokenTypes.NUMBER, TokenTypes.VARIABLE):

out.append(token)

elif token[1]['type'] == TokenTypes.NAMED\_FUNCTION:

op\_stack.append(token)

elif token[1]['type'] == TokenTypes.OPERATOR:

while ((len(op\_stack) > 0) #(stack[len(stack)][1]['type'] == TokenTypes.OPERATOR) # ?????

and (op\_stack[len(op\_stack)-1][1]['type'] != TokenTypes.LEFT\_PARENTHESIS)

and ((op\_stack[len(op\_stack)-1][1]['data']['precedence'] > token[1]['data']['precedence'])

or (op\_stack[len(op\_stack)-1][1]['data']['precedence'] == token[1]['data']['precedence'] and token[1]['data']['isLeftAssociative']))):

out.append(op\_stack.pop())

op\_stack.append(token)

elif token[1]['type'] == TokenTypes.LEFT\_PARENTHESIS:

op\_stack.append(token)

elif token[1]['type'] == TokenTypes.RIGHT\_PARENTHESIS:

while (op\_stack[len(op\_stack)-1][1]['type'] != TokenTypes.LEFT\_PARENTHESIS) or (len(op\_stack) == 0):

out.append(op\_stack.pop())

# If the stack runs out without finding a left parenthesis, then there are mismatched parentheses.

if len(op\_stack) == 0:

raise Exception('ShunYard error: no right parenthesis')

if op\_stack[len(op\_stack)-1][1]['type'] == TokenTypes.LEFT\_PARENTHESIS: # нужно ли?

op\_stack.pop()

if op\_stack[len(op\_stack)-1][1]['type'] == TokenTypes.NAMED\_FUNCTION:

out.append(op\_stack.pop())

while len(op\_stack) > 0:

out.append(op\_stack.pop())

#print\_tokens(out)

def shunting\_yard\_ast(tokens: t.List[t.Tuple[str, t.Dict]]) -> MathAST:

op\_stack = list()

out = list()

def op\_add(op: str):

b = out.pop()

if op[0] == '~':

out.append(Usub(b))

return

a = out.pop()

if op[0] == '+':

out.append(Add(a, b))

elif op[0] == '-':

out.append(Sub(a, b))

elif op[0] == '\*':

out.append(Mul(a, b))

elif op[0] == '/':

out.append(Div(a, b))

elif op[0] == '^':

out.append(Pow(a, b))

for token in tokens:

if token[1]['type'] in (TokenTypes.CONSTANT, TokenTypes.NUMBER, TokenTypes.VARIABLE):

if token[1]['type'] == TokenTypes.CONSTANT:

for x in constants:

if token[0] == x.\_\_name\_\_.lower():

out.append(x())

break

elif token[1]['type'] == TokenTypes.NUMBER:

out.append(Constant(float(token[0])))

elif token[1]['type'] == TokenTypes.VARIABLE:

out.append(Variable(token[0]))

#out.append(token)

elif token[1]['type'] == TokenTypes.NAMED\_FUNCTION:

op\_stack.append(token)

elif token[1]['type'] == TokenTypes.OPERATOR:

while ((len(op\_stack) > 0) #(stack[len(stack)][1]['type'] == TokenTypes.OPERATOR) # ?????

and (op\_stack[len(op\_stack)-1][1]['type'] != TokenTypes.LEFT\_PARENTHESIS)

and ((op\_stack[len(op\_stack)-1][1]['data']['precedence'] > token[1]['data']['precedence'])

or (op\_stack[len(op\_stack)-1][1]['data']['precedence'] == token[1]['data']['precedence'] and token[1]['data']['isLeftAssociative']))):

op\_add(op\_stack.pop()[0])

op\_stack.append(token)

elif token[1]['type'] == TokenTypes.LEFT\_PARENTHESIS:

op\_stack.append(token)

elif token[1]['type'] == TokenTypes.RIGHT\_PARENTHESIS:

while (op\_stack[len(op\_stack)-1][1]['type'] != TokenTypes.LEFT\_PARENTHESIS) or (len(op\_stack) == 0):

op\_add(op\_stack.pop()[0])

# If the stack runs out without finding a left parenthesis, then there are mismatched parentheses.

if len(op\_stack) == 0:

raise Exception('ShunYard error: no right parenthesis')

if op\_stack[len(op\_stack)-1][1]['type'] == TokenTypes.LEFT\_PARENTHESIS: # нужно ли?

op\_stack.pop()

if (len(op\_stack) > 0) and (op\_stack[len(op\_stack)-1][1]['type'] == TokenTypes.NAMED\_FUNCTION):

fun = op\_stack.pop()

a = out.pop()

for x in functions:

if fun[0] == x.\_\_name\_\_.lower():

out.append(x(a))

break

while len(op\_stack) > 0:

op\_add(op\_stack.pop()[0])

#print(out[0])

return out[0]

# def rpn\_to\_ast(tokens):

# stack = list()

# for token in tokens:

# if token[1]['type'] in (TokenTypes.CONSTANT, TokenTypes.NUMBER, TokenTypes.VARIABLE):

# stack.append(token)

# if token[1]['type'] in (TokenTypes.OPERATOR, TokenTypes.NAMED\_FUNCTION):

def parse(s: str) -> MathAST:

prepare = ("".join(s.split())).lower()

tokens = tokenize(prepare)

return shunting\_yard\_ast(tokens)

#shunting\_yard\_ast(tokenize('1+5-4\*a\*(3/x)^sin(3)'))

Листинг 5. math\_composer.py

from math\_ast import \*

def compose(ast: MathAST) -> str:

def wrap\_parethesis(t: MathAST) -> str:

if isinstance(t, LeafToken) or isinstance(t, SingleToken):

return compose(t)

else:

return '(' + compose(t) + ')'

if isinstance(ast, Pi):

return 'pi'

elif isinstance(ast, Exponenta):

return 'e'

elif isinstance(ast, SingleToken):

if isinstance(ast, Usub):

return '-' + wrap\_parethesis(ast.a)

else:

return '{}({})'.format(ast.\_\_class\_\_.\_\_name\_\_.lower(), compose(ast.a))

elif isinstance(ast, DoubleToken):

if isinstance(ast, Add): op = '+'

if isinstance(ast, Sub): op = '-'

if isinstance(ast, Mul): op = '\*'

if isinstance(ast, Div): op = '/'

if isinstance(ast, Pow): op = '^'

return wrap\_parethesis(ast.a) + op + wrap\_parethesis(ast.b)

elif isinstance(ast, Variable):

return ast.name

elif isinstance(ast, Constant):

return f'{ast.value:g}'

else:

raise Exception('Compose error: unknown node type: '+ast.\_\_class\_\_.\_\_name\_\_)

Листинг 6. simplifier.py

from math import pi

from calculator import calculate

from math\_ast import \*

from utils import \*

def simplify(ast: MathAST) -> MathAST:

if isinstance(ast, LeafToken):

return ast

if not is\_var\_dependant(ast):

return Constant(calculate(ast))

if isinstance(ast, SingleToken):

ast.a = simplify(ast.a)

if isinstance(ast, Usub) and isinstance(ast.a, Usub):

return ast.a.a

elif isinstance(ast, DoubleToken):

ast.a = simplify(ast.a)

ast.b = simplify(ast.b)

# if not is\_var\_dependant(ast):

# return Constant(calculate(ast))

if isinstance(ast, Add):

if isinstance(ast.a, Constant) and (ast.a.value == 0):

return ast.b

elif isinstance(ast.b, Constant) and (ast.b.value == 0):

return ast.a

elif isinstance(ast, Sub):

if isinstance(ast.a, Constant) and (ast.a.value == 0):

return Usub(ast.b)

elif isinstance(ast.b, Constant) and (ast.b.value == 0):

return ast.a

elif isinstance(ast, Mul):

if (isinstance(ast.a, Constant) and (ast.a.value == 0)) \

or (isinstance(ast.b, Constant) and (ast.b.value == 0)):

return Constant(0)

elif isinstance(ast.a, Constant) and (ast.a.value == 1):

return ast.b

elif isinstance(ast.b, Constant) and (ast.b.value == 1):

return ast.a

elif isinstance(ast, Div):

if isinstance(ast.a, Constant) and (ast.a.value == 0):

return Constant(0)

elif isinstance(ast.b, Constant) and (ast.b.value == 1):

return ast.a

elif isinstance(ast, Pow):

if isinstance(ast.b, Constant) and (ast.b.value == 0):

return Constant(1)

elif isinstance(ast.b, Constant) and (ast.b.value == 1):

return ast.a

return ast

Листинг 7. calculator.py

from math\_parser import parse

from math\_ast import \*

import math

def calculate(ast: MathAST, \*\*kwargs) -> float:

def calc(ast: MathAST) -> float:

if isinstance(ast, Constant):

return ast.value

elif isinstance(ast, Pi):

return math.pi

elif isinstance(ast, Exponenta):

return math.e

elif isinstance(ast, Sin):

return math.sin(calc(ast.a))

elif isinstance(ast, Cos):

return math.cos(calc(ast.a))

elif isinstance(ast, Tg):

return math.tan(calc(ast.a))

elif isinstance(ast, Ctg):

return math.cos(calc(ast.a))/math.sin(calc(ast.a))

elif isinstance(ast, Ln):

return math.log10(calc(ast.a))

elif isinstance(ast, Usub):

return -calc(ast.a)

elif isinstance(ast, Arcsin):

return math.asin(calc(ast.a))

elif isinstance(ast, Arccos):

return math.acos(calc(ast.a))

elif isinstance(ast, Arctg):

return math.atan(1/calc(ast.a))

elif isinstance(ast, Sqrt):

return math.sqrt(calc(ast.a))

elif isinstance(ast, Add):

return calc(ast.a) + calc(ast.b)

elif isinstance(ast, Sub):

return calc(ast.a) - calc(ast.b)

elif isinstance(ast, Mul):

return calc(ast.a) \* calc(ast.b)

elif isinstance(ast, Div):

return calc(ast.a) / calc(ast.b)

elif isinstance(ast, Pow):

return calc(ast.a) \*\* calc(ast.b)

elif isinstance(ast, Variable):

if ast.name in kwargs:

return kwargs[ast.name]

else:

raise Exception('Calculate error: no value provided for variable: '+ast.name)

else:

raise Exception('Calculate error: unknown MathAST node type')

return calc(ast)

if \_\_name\_\_ == "\_\_main\_\_":

print(calculate(parse('(4+6/5)-3.5'), x=4, a=3.5))

Листинг 8. derivative.py

from math\_ast import \*

from utils import \*

def differentiate(f: MathAST, dx: str) -> MathAST:

if isinstance(f, Differentiate): return differentiate(f.f, f.dx)

elif isinstance(f, Add): return Add(differentiate(f.a, dx), differentiate(f.b, dx))

elif isinstance(f, Sub): return Add(differentiate(f.a, dx), differentiate(f.b, dx))

elif isinstance(f, Div): return Div(Sub(Mul(differentiate(f.a, dx), f.b), Mul(f.a, differentiate(f.b, dx))), Pow(f.b, Constant(2)))

elif isinstance(f, Pow):

u = is\_depends\_on\_dx(f.a, dx)

v = is\_depends\_on\_dx(f.b, dx)

if u and not v:

return Mul(Mul(Pow(f.a, Sub(f.b, Constant(1))), differentiate(f.a, dx)), f.b) # u(x)^c, c=const

elif not u and v: # c^u(x), c=const

return Mul(Mul(Pow(f.a, f.b), differentiate(f.b, dx)), Ln(f.a))

elif not u and not v:

return Constant(0)

else:

return Mul(Pow(f.a, Sub(f.b, Constant(1))), Add(Mul(f.b, differentiate(f.a, dx)), Mul(Mul(f.a, Ln(f.a)), differentiate(f.b, dx))))

elif isinstance(f, Mul):

u = is\_depends\_on\_dx(f.a, dx)

v = is\_depends\_on\_dx(f.b, dx)

if u and not v:

return Mul(differentiate(f.a, dx), f.b)

elif not u and v: # c^u(x), c=const

return Mul(f.a, differentiate(f.b, dx))

elif not u and not v: # c^c, c=const

return Constant(0)

else:

return Add(Mul(differentiate(f.a, dx), f.b), Mul(f.a, differentiate(f.b, dx)))

elif isinstance(f, SingleToken):

if isinstance(f, Sin): d = Cos(f.a)

if isinstance(f, Cos): d = Usub(Sin(f.a))

if isinstance(f, Tg): d = Div(Constant(1), Pow(Cos(f.a), Constant(2)))

if isinstance(f, Ctg): d = Usub(Div(Constant(1), Pow(Sin(f.a), Constant(2))))

if isinstance(f, Abs): d = Div(f.a, Abs(f.a))

if isinstance(f, Ln): d = Div(Constant(1), f.a)

if isinstance(f, Sqrt): d = Div(Constant(1), Mul(Constant(2), Sqrt(f.a)))

if isinstance(f, Usub): d = Usub(differentiate(f.a, dx))

if isinstance(f, Arcsin): d = Div(Constant(1), Sqrt(Sub(Constant(1), Pow(f.a, Constant(2)))))

if isinstance(f, Arccos): d = Usub(Div(Constant(1), Sqrt(Sub(Constant(1), Pow(f.a, Constant(2))))))

if isinstance(f, Arctg): d = Div(Constant(1), Sub(Constant(1), Pow(f.a, Constant(2))))

if isinstance(f, Arcctg): d = Usub(Div(Constant(1), Sub(Constant(1), Pow(f.a, Constant(2)))))

if (is\_leaf(f.a)):

return d

else:

return Mul(d, differentiate(f.a, dx))

elif isinstance(f, Variable):

if (f.name == dx):

return Constant(1)

else:

return Constant(0)

elif isinstance(f, Constant):

return Constant(0)

elif isinstance(f, Pi) or isinstance(f, Exponenta):

return Constant(0)

else:

raise Exception('Derivative error: unknown MathAST node type')

def detect\_vars\_and\_diff(f: MathAST) -> MathAST:

v = detect\_vars(f)

if len(v) > 1:

raise Exception('Derivative error: function have 2 or more variables')

if len(v) == 1:

return differentiate(f, v[0].name)

else:

return differentiate(f, '')

Листинг 9. integral.py

from math\_ast import MathAST

from calculator import calculate

from utils import \*

class Quadrature:

"""Базовые определения для квадратурных формул"""

\_\_sum = 0.0

\_\_nseg = 1 # число отрезков разбиения

\_\_ncalls = 0 # считает число вызовов интегрируемой функции

\_\_var: str

def \_\_restart(func, x0, x1, nseg0, reset\_calls = True):

"""Обнуление всех счётчиков и аккумуляторов.

Возвращает интеграл методом трапеций на начальном разбиении"""

if reset\_calls:

Quadrature.\_\_ncalls = 0

Quadrature.\_\_nseg = nseg0

# вычисление суммы для метода трапеций с начальным числом отрезков разбиения nseg0

Quadrature.\_\_sum = 0.5 \* (calculate(func, \*\*{Quadrature.\_\_var: x0}) + calculate(func, \*\*{Quadrature.\_\_var: x1}))

dx = 1.0 \* (x1 - x0) / nseg0

for i in range(1, nseg0):

Quadrature.\_\_sum += calculate(func, \*\*{Quadrature.\_\_var: x0 + i \* dx})

Quadrature.\_\_ncalls += 1 + nseg0

return Quadrature.\_\_sum \* dx

def \_\_double\_nseg(func, x0, x1):

"""Вдвое измельчает разбиение.

Возвращает интеграл методом трапеций на новом разбиении"""

nseg = Quadrature.\_\_nseg

dx = (x1 - x0) / nseg

x = x0 + 0.5 \* dx

i = 0

AddedSum = 0.0

for i in range(nseg):

AddedSum += calculate(func, \*\*{Quadrature.\_\_var: x + i \* dx})

Quadrature.\_\_sum += AddedSum

Quadrature.\_\_nseg \*= 2

Quadrature.\_\_ncalls += nseg

return Quadrature.\_\_sum \* 0.5 \* dx

def trapezoid(func: MathAST, var: str, x0: float, x1: float, rtol = 1e-10, nseg0 = 1) -> float:

"""Интегрирование методом трапеций с заданной точностью.

rtol - относительная точность,

nseg0 - число отрезков начального разбиения"""

Quadrature.\_\_var = var

ans = Quadrature.\_\_restart(func, x0, x1, nseg0)

old\_ans = 0.0

err\_est = max(1, abs(ans))

while (err\_est > abs(rtol \* ans)):

old\_ans = ans

ans = Quadrature.\_\_double\_nseg(func, x0, x1)

err\_est = abs(old\_ans - ans)

return ans

def simpson(func: MathAST, var: str, x0: float, x1: float, rtol = 1.0e-10, nseg0 = 1) -> float:

"""Интегрирование методом парабол с заданной точностью.

rtol - относительная точность,

nseg0 - число отрезков начального разбиения"""

Quadrature.\_\_var = var

old\_trapez\_sum = Quadrature.\_\_restart(func, x0, x1, nseg0)

new\_trapez\_sum = Quadrature.\_\_double\_nseg(func, x0, x1)

ans = (4 \* new\_trapez\_sum - old\_trapez\_sum) / 3

old\_ans = 0.0

err\_est = max(1, abs(ans))

while (err\_est > abs(rtol \* ans)):

old\_ans = ans

old\_trapez\_sum = new\_trapez\_sum

new\_trapez\_sum = Quadrature.\_\_double\_nseg(func, x0, x1)

ans = (4 \* new\_trapez\_sum - old\_trapez\_sum) / 3

err\_est = abs(old\_ans - ans)

#print("Total function calls: " + str(Quadrature.\_\_ncalls))

return ans

def detect\_vars\_and\_simp(f: MathAST, x0: float, x1: float) -> MathAST:

v = detect\_vars(f)

if len(v) > 1:

raise Exception('Integral error: function have 2 or more variables')

if len(v) == 1:

return Quadrature.simpson(f, v[0].name, x0, x1)

else:

return Quadrature.simpson(f, '', x0, x1)

Листинг 10. gui.py

from calculator import calculate

from simplifier import simplify

from derivative import differentiate, detect\_vars\_and\_diff

from math\_parser import parse

from math\_composer import compose

from integral import detect\_vars\_and\_simp

from tkinter import StringVar, Text, ttk

from tkinter import messagebox

import tkinter as tk

import traceback

def solve\_diff():

vvod = str(formula\_diff.get())

try:

ast = parse(vvod)

ast = simplify(ast)

print('simplified:\n'+compose(ast))

deriv\_ast = detect\_vars\_and\_diff(ast)

print('derivative:\n'+compose(deriv\_ast))

deriv\_ast = simplify(deriv\_ast)

print('simplified derivative:\n'+compose(deriv\_ast))

vivod\_diff.delete(0,"end")

vivod\_diff.insert(0, compose(deriv\_ast))

except Exception as e:

traceback.print\_exc()

def solve\_integ():

vvod\_formula = str(formula\_integ.get())

vvod\_a = float(chislo\_a.get())

vvod\_b = float(chislo\_b.get())

try:

ast = parse(vvod\_formula)

ast = simplify(ast)

print('simplified:\n'+compose(ast))

integ = detect\_vars\_and\_simp(ast,vvod\_a,vvod\_b)

print('area under a curve:\n'+str(integ))

vivod\_integ.delete(0,"end")

vivod\_integ.insert(0, str(integ))

except Exception as e:

traceback.print\_exc()

root = tk.Tk()

root.geometry('550x230')

root.minsize(550, 230)

root.maxsize(550, 230)

root.title('Calculator')

notebook = ttk.Notebook(root)

notebook.pack(pady=10, expand=True)

frame1 = ttk.Frame(notebook, width=300, height=100)

frame2 = ttk.Frame(notebook, width=500, height=300)

frame1.pack(fill='both', expand=True)

frame2.pack(fill='both', expand=True)

notebook.add(frame1, text='Differentiation')

notebook.add(frame2, text='Integration')

tk.Label(frame1,text ="Input function:" ).grid(column = 1,

row = 2,)

name = StringVar()

formula\_diff = tk.Entry(frame1, width=70)

formula\_diff.grid(column= 2, row= 2)

button\_diff = tk.Button(frame1,text="Solve", width = 60, height=2, command=solve\_diff)

button\_diff.grid(row=4, column=2,)

tk.Label(frame1,text ="Output:").grid(column = 1,row = 5,)

name = StringVar()

vivod\_diff = tk.Entry(frame1, width=70, relief="ridge")

vivod\_diff.grid(column= 2, row= 5)

# tk.Label(frame2,text ='Use only "x" variable' ).grid(column = 1,row = 1,)

formula\_integ = tk.Entry(frame2, width=87)

formula\_integ.grid(column= 1, row= 1)

tk.Label(frame2,text ="a:" ).grid(column = 1,row = 2,)

chislo\_a = tk.Entry(frame2, width=87)

chislo\_a.grid(column= 1, row= 3)

tk.Label(frame2,text ="b:" ).grid(column = 1,row = 4,)

chislo\_b = tk.Entry(frame2, width=87)

chislo\_b.grid(column= 1, row= 5)

button\_integ = tk.Button(frame2,text="Solve", width = 60, height=2, command=solve\_integ).grid(column=1,row=6)

vivod\_integ = tk.Entry(frame2, width=70, relief="ridge")

vivod\_integ.grid(column= 1, row= 7)

root.mainloop()

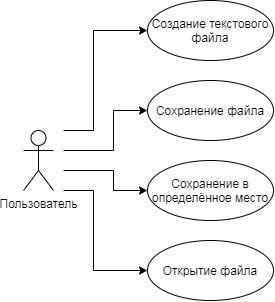
# Раздел 2. Проектирование и разработка программы «Блокнот».

## Построение UML-диаграмм к программе «Блокнот»

В данном разделе содержится диаграмма прецедентов для приложения, на которой показаны возможные функциональные отношения.

Рисунок 5.

Диаграмма прецедентов.



## Разработка программы «Блокнот»

Листинг 2. “Form1.cs”.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace NotepadWF

{

public partial class Form1 : Form

{

public string filename;

public bool isFileChanging;

public Form1()

{

InitializeComponent();

Init();

}

public void Init()

{

filename = "";

isFileChanging = false;

}

public void CreateNewDocument(Object sender, EventArgs e)

{

textBox1.Text = "";

filename = "";

}

public void OpenFile(Object sender, EventArgs e)

{

openFileDialog1.FileName = "";

if(openFileDialog1.ShowDialog() == DialogResult.OK)

{

try

{

StreamReader sr = new StreamReader(openFileDialog1.FileName);

textBox1.Text = sr.ReadToEnd();

filename = openFileDialog1.FileName;

}

catch

{

MessageBox.Show("Не возможно открыть файл!");

}

}

}

public void SaveFile(string \_filename)

{

if(\_filename == "")

{

if(saveFileDialog1.ShowDialog() == DialogResult.OK)

{

\_filename = saveFileDialog1.FileName;

}

}

try

{

StreamWriter sw = new StreamWriter(\_filename);

sw.Write(textBox1.Text);

sw.Close();

filename = \_filename;

isFileChanging = false;

}

catch

{

MessageBox.Show("Не возможно сохранить файл!");

}

}

public void Save(Object sender, EventArgs e)

{

SaveFile(filename);

}

public void SaveAs(Object sender, EventArgs e)

{

SaveFile("");

}

private void OnTextChanged(object sender, EventArgs e)

{

if (!isFileChanging)

{

isFileChanging = true;

this.Text = "\*" + this.Text;

}

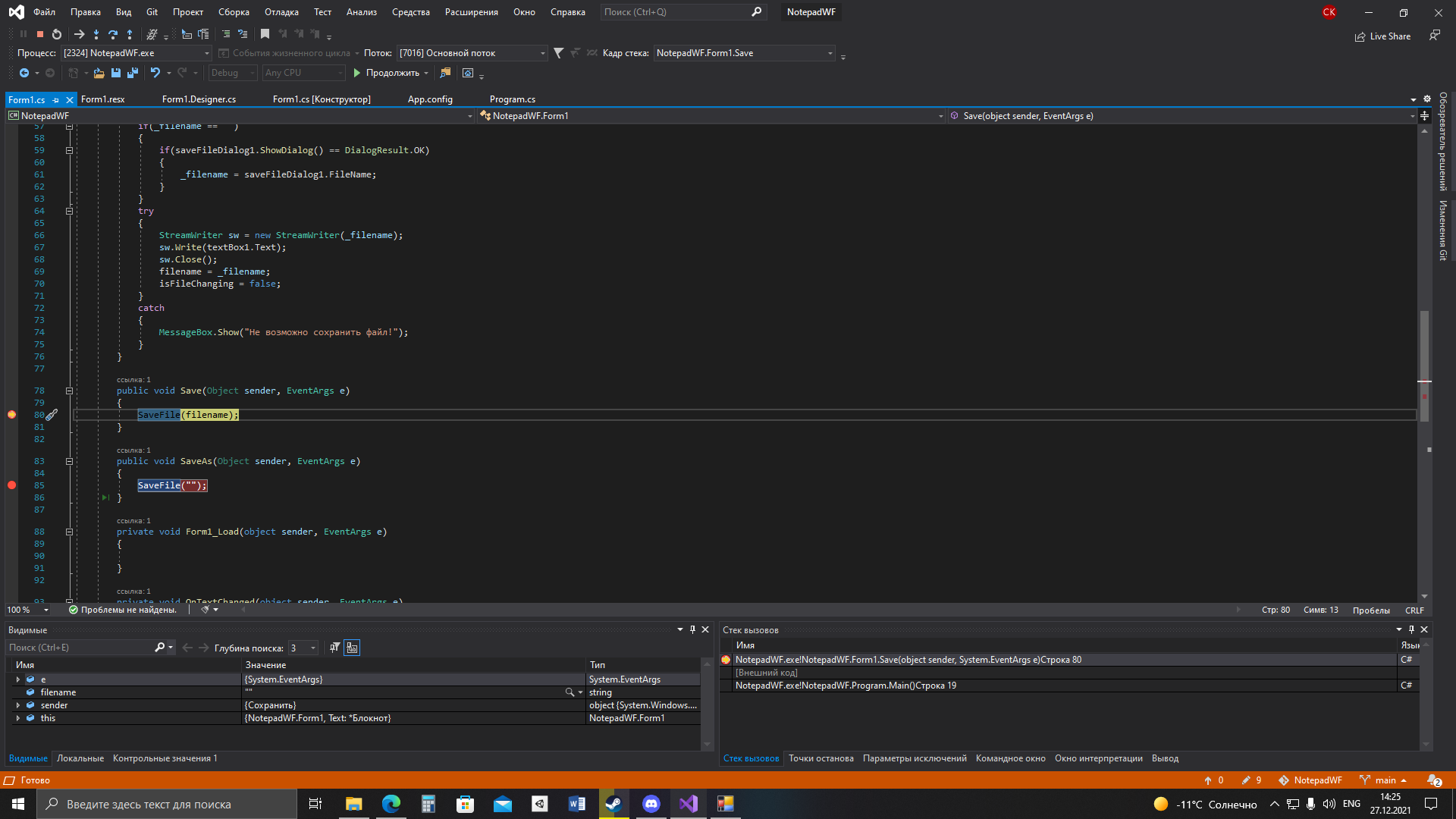
}

}

}

Рисунок 6.

Пример кода.



Расположение кнопок.

1. Примером для удобства расположения кнопок стал стандартный Microsoft Notepad.

Рисунок 7.

Дизайн приложения.

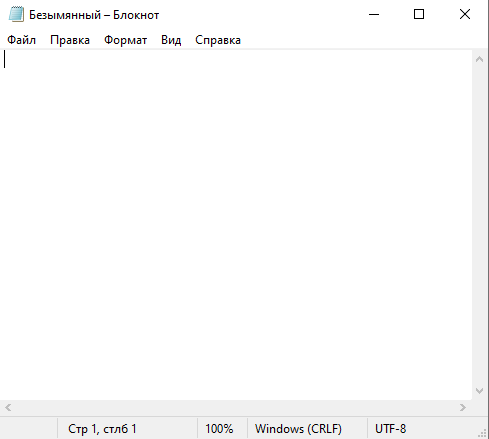
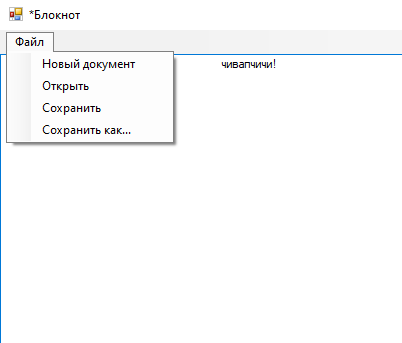


Рисунок 8.

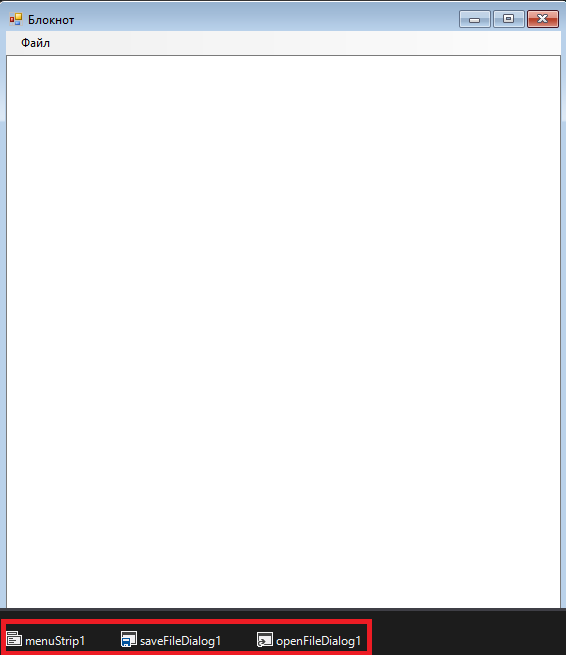
Реализация кнопки файл.



1. Кнопка “Файл” реализована с помощью компонента menuStrip
2. Функции у кнопок “Сохранить” и “Открыть” реализованы с помощью компонентов saveFileDialog и openFileDialog соответственно

Рисунок 9.

Компоненты menuStrip, saveFileDialog, openFileDialog.



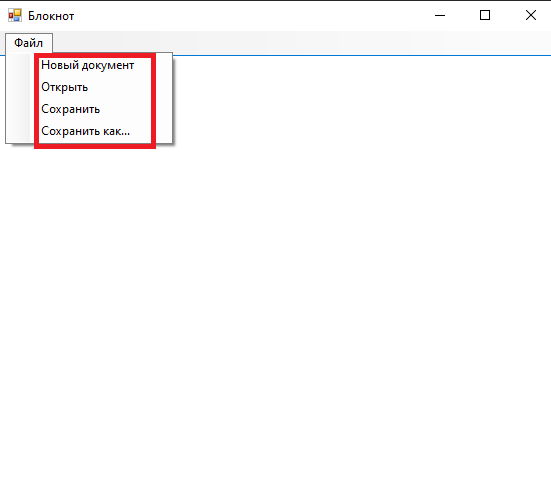
Взаимодействие с кнопками. При нажатии на кнопку “Новый документ” создается новый файл.

При нажатии на кнопку “Открыть” можно открыть существующий документ, созданный программой или другим текстовым редактором. При нажатии на кнопку “Сохранить” можно сохранить документ.

При нажатии на кнопку “Сохранить как…” можно сохранить документ в удобной для пользователя директории.

Рисунок 10.

Кнопки взаимодействия.



# Дерево файлов

Серийный номер тома: 2AF7-C78A

Рисунок 12.

Структура папок



# Заключение.

В результате выполнения приложений для учебной практики были разработаны, калькулятор и блокнот, данные наработки имеют потенциал для будущих улучшений. Разработанные программы были выполнены в соответствии с требованиями технического задания.