Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

**Лабораторная работа №2**

**Дисциплина «Символьные вычисления»**

**Вариант 21**

**Выполнил:**

Съестов Дмитрий Вячеславович

Группа P3317

**Преподаватель:**

Кореньков Юрий Дмитриевич

Санкт-Петербург

2018

**Задание**

Изучить основы функционирования символьных методов вычисления.

Реализовать заданный алгоритм над символьной формой выражения по вариантам с помощью произвольного языка программирования, не основываясь на существующих системах символьных вычислений.

Порядок выполнения:

• Проанализировать задействуемые в задании предметные области, произвести их декомпозицию, выявить сущности и разработать архитектуру решения

• Реализовать необходимые структуры данных и модели

• Реализовать соответствующие варианту задания методы загрузки и сохранения данных

• Реализовать соответствующий заданию алгоритм

• Оформить отчет по работе и продемонстрировать преподавателю для защиты вместе с демонстрацией разработанного решения

Входной формат: Eqn

Алгоритм: интегрирование

Выходной формат: AsciiMath

**Интегрирование**

function \_integrate(n, x)

@match n begin

0 => 0

1 => x

n::Number => Expr(:call, :\*, n, x)

\_ => tableIntegral(n, x)

end

end

function \_integrate(ex::Expr, x)

if ex.head ≠ :call

ArgumentError("ex must be a call expression") |> throw

end

op = ex.args[1]

@match op begin

:+ || :- => Expr(:call, op, (a -> integrate(a, x)).(ex.args[2:end])...)

:\* || :/ =>

@match consts(ex) begin

(0, \_) => 0

(1, ex) => tableIntegral(ex, x)

(n, ex) => Expr(:call, :\*, n, tableIntegral(ex, x))

end

\_ => tableIntegral(ex, x)

end

end

function tableIntegral(ex::Value, x)

dx = Expr(:call, :d, x)

int = Expr(:call, :\*, ex, dx)

for rule ∈ integralTable

pat = Expr(:call, :\*, rule.pat, :(d(x)))

match = matchex(pat, int; allow\_ex = true, exact = false)

if match ≠ nothing && rule.check(match)

return rewrite(ex, rule.pat, rule.rpat)

end

end

nothing

end

**Основная программа**

e = ℯ

oo = Inf

eqnRegex = let eqnexp(name) = string("(?<$(name)>", raw"(?:\{.\*\})|(?:[^\s]+))"),

sub\_then\_sup = string("(?:sub\\s+", eqnexp("a1"), "\\s+sup\\s+", eqnexp("b2"), ")"),

sup\_then\_sub = string("(?:sup\\s+", eqnexp("b1"), "\\s+sub\\s+", eqnexp("a2"), ")"),

indices = "(?:(?:$(sub\_then\_sup)|$(sup\_then\_sub))\\s+)?"

string("^\\s\*int\\s+", indices, eqnexp("ex"), "\$") |> Regex

end

eqnRules = Dict(

"{" => "(",

"}" => ")",

r"\s+ sup \s+"x => "^",

r"\s+ over \s+"x => "/"

)

function eqnToJulia(ex)

for rule ∈ eqnRules

ex = replace(ex, rule)

end

Meta.parse(ex)

end

function calculate(int::Expr, bounds)

@eval f(x) = $int

f(bounds[2]) - f(bounds[1])

end

calculate(int::Expr, bounds::Nothing) = int

calculate(::Nothing, \_) = error("неизестный интеграл.")

function wrapInf(x)

x == Inf && return :oo

x == -Inf && return :(-oo)

x

end

wrap(ex::Expr) = "($(walk(wrapInf, ex)))"

wrap(x) = "$(wrapInf(x))"

try

print("Введите выражение Eqn: \n> ")

int = readline()

m = match(eqnRegex, int)

m==nothing && error("некорректный ввод.")

a, b = defaultTo(m["a1"], m["a2"]), defaultTo(m["b1"], m["b2"])

if (a, b) == (nothing, nothing)

bounds = nothing

else

α, β = eqnToJulia(a), eqnToJulia(b)

bounds = eval(α), eval(β)

end

ex = eqnToJulia(m["ex"])

integral = integrate(ex, :x)

result = calculate(integral, bounds)

if bounds ≠ nothing

ascii\_bounds = " \_$(wrap(bounds[1])) ^$(wrap(bounds[2]))"

result\_rounded = round(result \* 1000) / 1000

eq\_sign = result\_rounded == result ? "=" : "~~"

print(string("int", ascii\_bounds, "$(wrap(ex))dx $(eq\_sign) ", result\_rounded))

else

print(string("int", "$(wrap(ex))dx = ", result))

end

catch e

if e isa Meta.ParseError

println("Ошибка парсинга выражения: $(e.msg)")

elseif e isa ErrorException

println("Ошибка: $(e.msg)")

else

rethrow()

end

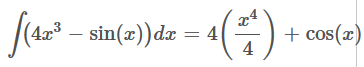
end

print("\nНажмите любую клавишу")

readline()

nothing

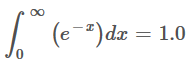
**Примеры**

Неопределённый интеграл

int {4x^3 - sin(x)}

Определённый интеграл

int sub -3 sup 10 {sin(x)}

Несобственный интеграл

int sub 0 sup oo {e sup -x}

**Вывод:** я реализовал интегрирование с помощью сопоставления выражения с образцом. Такой способ вполне для табличных интегралов, не учитывающих коммутативность выражений и другие тождества (например, *sin2x + cos2x* ⬄1). Для полноценной системы компьютерной алгебры требуются гораздо более сложные алгоритмы.