МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и Структуры Данных»
Тема: Иерархический список

Студент гр. 9304 _____ Тиняков С.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить структуру данных иерархический список и реализовать его на языке программирования C++.

Задание.

Вариант 27.

Символьное дифференцирование алгебраического выражения, рассматриваемого как функция от одной из переменных. На входе выражение в виде иерархического списка и переменная, по которой следует дифференцировать. На выходе – производная исходного выражения. После дифференцирования возможно упрощение выражения. Набор операций (функций), которые могут входить в выражение: +, -, *, /, sin(), cos(), упрощать не требуется

Выполнение работы.

Формат входных данных: префиксная запись выражения и переменная дифференцирования. Общий шаблон выражения: (<оперцаия> (<выражение1>) (<выражение2>)). Если операция явялется унарной, то вторая скобка должна отсутствовать. Всё выражение также должно находится в скобках. Выражение *(a+(-b*c))* будет выглядеть следующим образом: *(+ (a) (* (-(b)) (c)))*. Операции + и - могут быть как бинарными, так и унарными. Все пробельные символы(а также табуляции и переноса строки) игнорируются. После выражения должна следовать переменная для дифференцирования. Всё, что следует после, игнорируется. На выходе программа выдаёт строку, содержащую продифференцированное выражение. Программа работает со стандартными потоками ввода и ввыода. Несколько примеров. Выражение *cos(a+c*d)* будет выглядеть следующим образом: *(cos(+(a)(*(c)(d))))*, графическое представление изображено на рис.1.

Выражение *cos(256*sin(-r))* будет выглядеть так: *(cos(*(256)(sin(-(r)))))*, графическое представление изображено на рис.2.

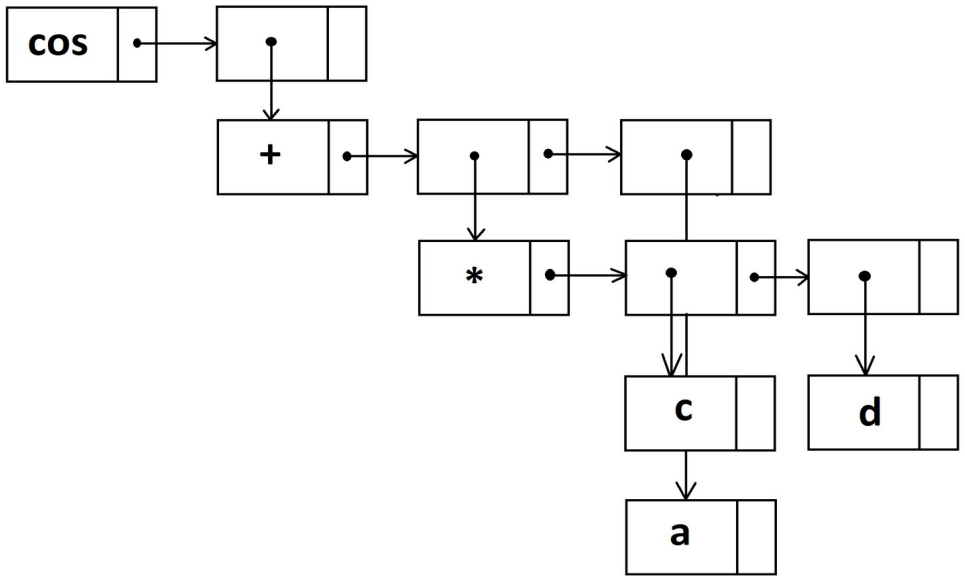

Рисунок 1 — Графическое представление выражения *cos(a+c*d)*

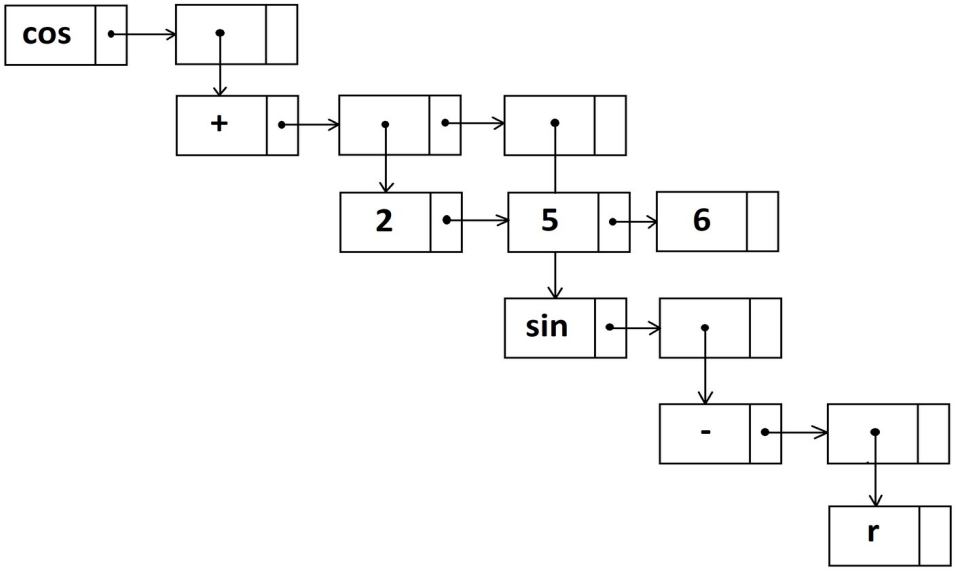

Рисунок 2 — Графическое представление выражения *cos(256*sin(-r))*

Работа программы состоит из двух частей: считывание выражения и дифференцирование. Оба алгоритма рекурсивны. Разберём алгоритм считывания выражения. Возможны два случая: на входе атом(число, переменная, константа) или на входе выражение. Если атом, то он возвращается. Отдельно надо упомянуть число, потому что оно хранится в виде списка, где все цифры находятся по порядку. Т.е. число *123* будет хранится как три атома: *1,2,3*. Если же на входе выражение, то сначала считывается операция и заносится в список, как атом, затем в следующий элементы списка при помощи рекурсивного испльзования данного алгоритма заносится выражения, которые следуют за оперцией. Алгоритм дифференцирования устроен почти также. Для данного алгоритма атом является тривиальным случаем выражения. Берётся выражение и для каждого выражения находящегося в нём вызывается данный алгоритм. Операция определяет по какому правилу брать производную.

Главной структурой является *Node*. Она хранит в себе указатель на следующий элемент, а также вариант из двух: указатель на *Node* или другой тип, который определяется шаблоном. Лямбда-функция *GetExpr* считывает выражение из входного потока и возвращает указатель на начало списка. Если ввод некорректен, то выбрасывается исключение *std::logic_error* с сообщением *„Invalid input“*. Лямбда-функция *DerivedExpr* вычисляет производную переданного выражения. Если передано некорректно построенно выражение, то выбросится исключение *std::logic_error* с сообщением *„List is broken“*. Также в *DerivedExpr* определена лямбда-функция *PrintExpr*, которая выводит выражение в стандартный поток вывода. Так как операции +, -, *, / можно хранить как символы, а косинус и синус нет, то для них соответственно используются символы *„&“* и *„$“*.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(+(a)(b)) a	(1)+(0)	Тест проверяет, что программа правильно считает производную суммы
2.	(-(a)(b)) b	(0)-(1)	Тест проверяет, что программа правильно считает производную разности
3.	(*(c)(d)) r	((0)*(d)+(c)*(0))	Тест проверяет, что программа правильно считает производную если переменной, по которой дифференцируют, нет в выражении
4.	(*(c)(d)) c	((1)*(d)+(c)*(0))	Тест проверяет, что программа правильно считает производную произведения
5.	(/(t)(d))t	((1)*(d)-(t)*(0))/(d)^2	Тест проверяет, что программа правильно считает производную частного
6.	(cos(*(45)(x)))x	-sin((45)*(x))*(((0)*(x)+(45)*(1)))	Тест проверяет, что программа правильно считает производную косинуса
7.	(sin(/(x)(45))) x	cos((x)/(45))*(((1)*(45)-(x)*(0))/(45)^2)	Тест проверяет, что программа правильно считает производную синуса
8.	(-(-(+(-(+(-(-(-(-(666)))))))))) y	-(-(+(-(+(-(-(-(0))))))))	Тест проверяет, что программа правильно работает с унарными видами операций + и -

Выводы.

Была изучена структура данных иерархический список и её реализация на языке программирования C++.

Была разработанна программа, которая считывает выражение и переменную и дифференцирует его по этой переменной. Структура для хранения списка была реализованна при помощи умных указателей и *std::variant*. Также в программе использовались лямбда-функции. Для проверки правильности работы программы были сделаны тесты.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab2.cpp

```
#include<iostream>
#include<memory>
#include<variant>
#include<cctype>

template<typename base>
class Node{
using NodePtr = std::shared_ptr<Node>;
public:
    NodePtr next {nullptr};
    std::variant<NodePtr, base> value;
};

int main(){
    std::istream& input = std::cin;
    std::ostream& output = std::cout;

    // cos == &
    // sin == $
                auto    GetExpr    =    [&input](auto&&    GetExpr)-
>std::shared_ptr<Node<char>>{
            char c;
            input >> c;
            if(c != '(') throw std::logic_error("Invalid input\n");
            auto head = std::make_shared<Node<char>>();
            input >> c;
               if(input.eof()) throw std::logic_error("Invalid input\
n");
            switch(c){
                case '+':
                case '-':
                case '*':
                case '/':
                    {
                        head->value = c;
                        head->next = std::make_shared<Node<char>>();
                        head->next->value = GetExpr(GetExpr);
                         if((c == '-' || c == '+') && input.peek() !=
EOF && input.peek() == ')'){
                            input >> c;
                            return head;
                        }
                                                    head->next->next   =
std::make_shared<Node<char>>();
                        head->next->next->value = GetExpr(GetExpr);
                        input >> c;
                                                if(input.eof())  throw
std::logic_error("Invalid input\n");
                        if(c == ')') return head;
                        throw std::logic_error("Invalid input\n");
```

```
                    }
                case 'c':
                    input >> c;
                      if(input.eof()) throw std::logic_error("Invalid
input\n");
                    if(c == 'o'){
                        input >> c;
                                                if(input.eof())  throw
std::logic_error("Invalid input\n");
                        if(c == 's'){
                            head->value = '&';
                                                        head->next  =
std::make_shared<Node<char>>();
                            head->next->value = GetExpr(GetExpr);
                            input >> c;
                                   if(input.eof() || c != ')') throw
std::logic_error("Invalid input\n");
                            return head;
                         }else throw std::logic_error("Invalid input\
n");
                    }else if(c == ')'){
                        head->value = 'c';
                        return head;
                    } else throw std::logic_error("Invalid input\n");
                case 's':
                    input >> c;
                      if(input.eof()) throw std::logic_error("Invalid
input\n");
                    if(c == 'i'){
                        input >> c;
                                                if(input.eof())  throw
std::logic_error("Invalid input\n");
                        if(c == 'n'){
                            head->value = '$';
                                                        head->next  =
std::make_shared<Node<char>>();
                            head->next->value = GetExpr(GetExpr);
                            input >> c;
                                   if(input.eof() || c != ')') throw
std::logic_error("Invalid input\n");
                            return head;
                         }else throw std::logic_error("Invalid input\
n");
                    }else if(c == ')'){
                        head->value = 's';
                        return head;
                    } else throw std::logic_error("Invalid input\n");
                case ')':
                    throw std::logic_error("Invalid input\n");
                default:
                    if(isdigit(c)){
                        auto cur = head;
                        head->value = c;
                        while(isdigit(c)){
                            input >> c;
```

```
                                                    if(input.eof())  throw
std::logic_error("Invalid input\n");
                                if(!isdigit(c)) break;
                                                                cur->next  =
std::make_shared<Node<char>>();
                                cur = cur->next;
                                cur->value = c;
                            }
                             if(c != ')') throw std::logic_error("Invalid
input\n");
                            return head;
                        }
                        if(isalpha(c)){
                            head->value = c;
                            input >> c;
                                                    if(input.eof())  throw
std::logic_error("Invalid input\n");
                            if(c == ')') return head;
                              else throw std::logic_error("Invalid input\
n");
                        }else throw std::logic_error("Invalid input\n");
                }
            };


                auto  DerivedExpr  =  [&output](std::shared_ptr<Node<char>>
head, char var, auto&& DerivedExpr)->void{
                    auto  PrintExpr  =  [&output](std::shared_ptr<Node<char>>
head, auto&& PrintExpr)->void{
                     if(!std::holds_alternative<char>(head->value)) throw
std::logic_error("List is broken\n");
                    char c = std::get<char>(head->value);
                    if(isdigit(c)){
                        while(head){
                            output << std::get<char>(head->value);
                            head = head->next;
                        }
                    } else if(isalpha(c)){
                        output << c;
                    } else if(head->next->next){
                        output << "(";
                         PrintExpr(std::get<std::shared_ptr<Node<char>>>(h
ead->next->value), PrintExpr);
                        output << ")" << c << "(";
                         PrintExpr(std::get<std::shared_ptr<Node<char>>>(h
ead->next->next->value), PrintExpr);
                        output << ")";
                    } else {
                        if(c == '$') output << "sin(";
                        else if(c == '&') output << "cos(";
                        else if(c == '+' || c == '-') output << c << "(";
                        else throw std::logic_error("List is broken\n");
                         PrintExpr(std::get<std::shared_ptr<Node<char>>>(h
ead->next->value), PrintExpr);
                        output << ")";
                    }
```

```
            };
                 if(!std::holds_alternative<char>(head->value))  throw
std::logic_error("List is broken\n");
            char c = std::get<char>(head->value);
            if(c == var) output << '1';
            else if(isalpha(c)) output << '0';
            if(isdigit(c)) output << "0";
            switch(c){
                case '+':
                case '-':
                    if(head->next->next){
                        output << "(";
                         DerivedExpr(std::get<std::shared_ptr<Node<cha
r>>>(head->next->value), var, DerivedExpr);
                        output << ")" << c << "(";
                         DerivedExpr(std::get<std::shared_ptr<Node<cha
r>>>(head->next->next->value), var, DerivedExpr);
                        output << ")";
                    }else{
                        output << c << "(";
                         DerivedExpr(std::get<std::shared_ptr<Node<cha
r>>>(head->next->value), var, DerivedExpr);
                        output << ")";
                    }
                    break;
                case '*':
                case '/':
                    output << "((";
                     DerivedExpr(std::get<std::shared_ptr<Node<char>>>
(head->next->value), var, DerivedExpr);
                    output << ")*(";
                     PrintExpr(std::get<std::shared_ptr<Node<char>>>(h
ead->next->next->value), PrintExpr);
                    if(c == '*') output << ")+(";
                    else output << ")-(";
                     PrintExpr(std::get<std::shared_ptr<Node<char>>>(h
ead->next->value), PrintExpr);
                    output << ")*(";
                     DerivedExpr(std::get<std::shared_ptr<Node<char>>>
(head->next->next->value), var, DerivedExpr);
                    output << "))";
                    if(c == '/'){
                        output << "/(";
                         PrintExpr(std::get<std::shared_ptr<Node<char>
>>(head->next->next->value), PrintExpr);
                        output << ")^2";
                    }
                    break;
                case '&':
                case '$':
                    if(c == '&') output << "-sin(";
                    else output << "cos(";
                     PrintExpr(std::get<std::shared_ptr<Node<char>>>(h
ead->next->value), PrintExpr);
                    output << ")*(";
```

```
                        DerivedExpr(std::get<std::shared_ptr<Node<char>>>
(head->next->value), var, DerivedExpr);
                        output << ")";
                };
            };

            try{
                std::shared_ptr<Node<char>> head = GetExpr(GetExpr);
                if(head == nullptr){
                    output << "Error in reading\n";
                    return 1;
                }
                char var;
                input >> var;
                if(!isalpha(var)){
                     output << "Wrong variable for derived: \'" << var <<
"\'\n";
                    return 1;
                }
                DerivedExpr(head, var, DerivedExpr);
                output << "\n";
            }catch(std::exception& e){
                std::cout << e.what();
                return 1;
            }
            return 0;
        }
```

Название файла: Makefile

```
.PHONY: all

all: run_tests

lab2: Source/lab2.cpp
	g++ Source/lab2.cpp -std=c++17 -o lab2

run_tests: lab2
	python3 test.py
```

Название файла: test.py

```
import unittest
import subprocess
import os
import filecmp

class TestParamAnalyzer(unittest.TestCase):

    cwd = os.getcwd()
    test_dir = './Tests/'
    tests = []

    @classmethod
```

```
        def setUpClass(self):
            files = os.listdir(self.test_dir)
            for f in files:
                if(f.endswith('.in')):
                    out = f[:f.rfind('.')] + ".out"
                    if(files.count(out) > 0):
                                self.tests.append([self.test_dir + f,
self.test_dir + out])

        def test_all(self):
            print('Start Testing...')
            out = 'output.test'
            for test in self.tests:
                with open(test[0], 'r') as f_in:
                    with open(out, 'w') as f_out:
                                p = subprocess.run(['./lab2',], cwd =
self.cwd, stdin = f_in, stdout = f_out)
                with open(test[0], 'r') as f_in:
                    print('Input: ',f_in.read(), sep='')
                with open(out, 'r') as f_out:
                    print('Output: ', f_out.read(), sep='')
                self.assertTrue(filecmp.cmp(out, test[1]))
            #    if os.path.isfile(out):
            #        os.remove(out)
            print('End Testing')

        def tearDown(self):
            out = 'output.test'
            if os.path.isfile(out):
                os.remove(out)


    if __name__ == "__main__":
        unittest.main()
```