

**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: Иерархические списки**

Студент гр. 9304

Сорин А.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

## Цель работы.

Узнать о иерархическом списке и его использовании в практике.

## Задание.

Пусть выражение (логическое, арифметическое, алгебраическое\*) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ( ( ) ), либо в постфиксной форме ( ) ). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (\* b (- c))) или (OR a (AND b (NOT c))). В задании даётся один из следующих вариантов требуемого действия с выражением: проверка синтаксической корректности, упрощение (преобразование), вычисление. Пример упрощения: (+ 0 (\* 1 (+ a b))) преобразуется в (+ a b). В задаче вычисления на входе дополнительно задаётся список значений переменных ( (x1 c1) (x2 c2) ... (xk ck) ), где  $x_i$  – переменная, а  $c_i$  – её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно - состав операций), вариант действия и форма записи. Всего 9 заданий.

\* - здесь примем такую терминологию: в арифметическое выражение входят операции +, -, \*, /, а в алгебраическое – +, -, \* и дополнительно некоторые функции. Здесь реализовано задание 21 арифметическое, вычисление, постфиксная форма

## **Формат входных и выходных данных.**

На вход подается постфиксное выражение и значения переменных.  
Например:

$(a \ b + (2,4 \ 1 \ * ) / ) (a \ 12 ), (b \ 2 )$

На выходе также будет это выражение, а потом результат вычисления.

## **Выполнение работы.**

Для выполнения работы был создан класс иерархического списка `h_list`. Все классы в работе шаблонные. Так как в списке нужно хранить как числа, так и операции с переменными, был создан класс `VarNum`, хранящий информацию о том, что хранится, и саму информацию.

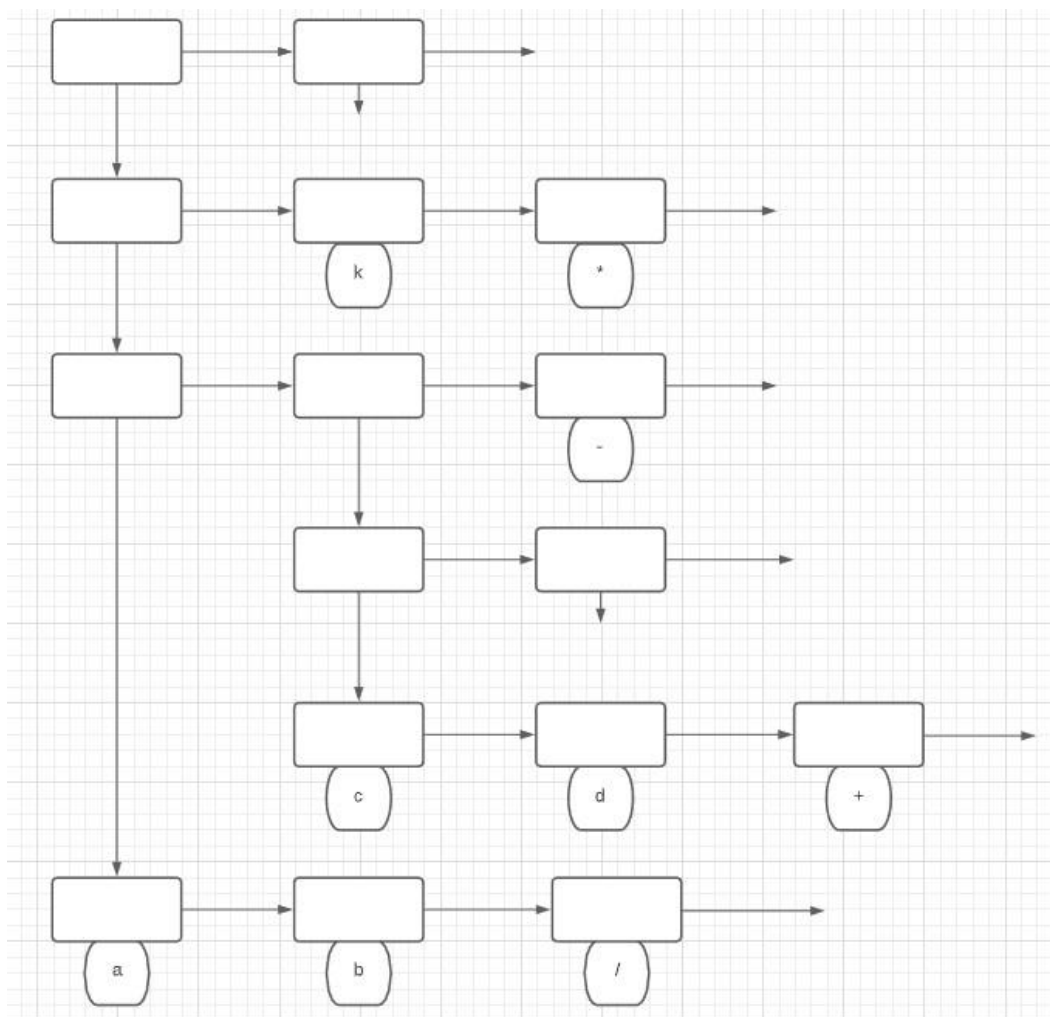
Иерархический список был реализован через умный указатель на него. В нем есть 2 поля. Указатель на следующий элемент `next` и `value`, который реализован через `std::variant` и может быть либо указателем на `h_list`, либо `VarNum`. Также есть метод для добавления `next`.

Также был создан класс `calc`. У него есть 2 приватных поля – умный указатель на иерархический список `H_List` и массив `VarValue` – массив со значениями переменных. У класса `calc` есть 1 конструктор и 2 публичных метода – `ReadExpr`, который вызывает 2 приватных метода `ReadExprRec` и `ReadVarValue`, и `CalcExpr`, который возвращает значение `CalcExprRec`. Также есть 8 приватных методов. Метод `H_ListToValueOfRoot` создает умный указатель и подвешивает на него выражение, которое получает на вход. Метод `ReadNumber` получает на вход цифру и считывает число, которое он возвращает. Метод `ReadNumberToH_List` использует `ReadNumber` и записывает число в список. Метод `ReadVar` считывает переменную и записывает ее в список. Метод `ReadOper` считывает операцию и записывает ее в список. Метод `ReadExprRec` считывает выражение и записывает в список. Это происходит следующим образом: есть три различных состояния. Для первых двух состояний можно считать число или переменную, которые запишутся в поле `value` и произойдет

переход к следующему полю списка. Также можно открыть скобки, в результате чего для текущего узла списка `value` тоже будет списком и для него рекурсивно вызовется `ReadExprRec`, после чего произойдет переход к следующему элементу списка. При всех этих действиях текущее состояние увеличится. Если в данный момент состояние 1, то значит можно закрыть скобки, в результате чего работа функции завершится или откатится назад в рекурсии. Если состояние равно 2, то можно только ввести операцию. Также состояние вернется к единице. Метод `ReadVarValue` считывает значения для переменных. Он записывает значение в элемент массива `VarValue` с номером, равным символу переменной. Затем он либо вызывает себя рекурсивно, либо заканчивает работу. Последний метод – `CalcExprRec`. Он считывает значение выражения в списке. Если список пустой, то метод выкидывает `invalid_argument`. Если в списке 1 элемент, то если это просто переменная или число, то метод его возвращает, а если там выражение, то функция вызывается рекурсивно. В случае если не 1 элемент, то так же записывается число в переменную, затем так же в следующую. После этого проверяется операция и применяется. После чего возвращается результат.

Пример иерархического списка:

Для выражения  $( a b / ( c d + ) - k * )$



### **Тестирование.**

Тестирование проводится при помощи программы на с++. При этом из файла считывается строка с выражением, затем применяются методы, чтобы посчитать результат. После этого, посчитанный результат сравнивается с правильным и выводится, пройден тест, или нет.

Результаты тестирования в приложении Б.

### **Выводы.**

Стало известно о иерархическом списке и его использовании в практике.

## **ПРИЛОЖЕНИЕ А**

### **ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: main.cpp

```
#include <stdexcept>
```

```
#include <string>
```

```
#include "Calc.h"
```

```
int main() {
```

```
try
```

```
{
```

```

    h_list<double> L;

    calk<double> C(L);

    double Res;

    std::string Str;

    if (!std::getline(std::cin, Str))

        throw std::runtime_error("Error while reading from stream");

    std::stringstream Stream(Str);

    C.ReadExpr(Stream);

    Res = C.CalcExpr();

    std::cout << '\n' << Res << '\n';

    system("pause");

}

catch (const std::exception& Error)

{

    std::cout << '\n' << Error.what();

}

return 0;

}

```

Название файла: h\_list.h

```
#ifndef __H_LIST
```

```
#define __H_LIST
```

```
#include <iostream>
```

```
#include <variant>
```

```
template <typename base>
```

```
class VarNum {
```

```
public:
```

```
bool IsVar = false;
```

```
bool IsOp = false;
```

```
char Var = 0;
```

```
base Num = 0;
```

```
};
```

```
template <typename base>
```

```
class h_list {
```

```
using h_list_ptr = std::shared_ptr<h_list>;
```

```
public:
```

```
h_list_ptr next{ nullptr };
```

```
std::variant<h_list_ptr, VarNum<base>> value;
```

```
void AddNext(void) {
```

```
    next = std::make_shared<h_list>();
```

```
}
```

```
};
```

```
#endif // __H_LIST
```

```
Название файла: calc.h
```

```
#ifndef __CALC
```

```
#define __CALC
```



```

#include "h_list.h"

#include <stdexcept>

#include <conio.h>

#include <sstream>


template <typename base>

class calk {

    std::shared_ptr<h_list<base>> H_List;

    base *VarValue['z' + 1];

    void H_ListToValueOfRoot(std::shared_ptr<h_list<base>> &L,
std::shared_ptr<h_list<base>>& save) {

        std::shared_ptr<h_list<base>> Root = std::make_shared<h_list<base>>();

        Root->value = save;

        save = Root;

        L = save;

    }

    base ReadNumber(char k, std::stringstream& Stream) {

        base Num = 0;

        if (k == '0')

            {

```

```

std::cout << '0';

if (!Stream.get(k))

    throw std::invalid_argument("Error while entering
expression");

if (k == ',')
{
    std::cout << ',';

    int I = 0;

    if (!Stream.get(k))

        throw std::invalid_argument("Error while entering
expression");

    if (k < '0' || k > '9')

        throw std::invalid_argument("Error while entering
expression");

    Num += 0.1 * ((base)k - '0');

    I++;

    std::cout << k;

    while (1)
    {

        if (!Stream.get(k))

            throw std::invalid_argument("Error while
entering expression");

        if (k >= '0' && k <= '9')

            {

```

```

        base deg = 0.1;

        for (int i = 0; i < I; i++)

            deg /= 10;

        Num += deg * ((base)k - '0');

        I++;

        std::cout << k;

    }

    else if (k == ' ')

        return Num;

    else

        throw std::invalid_argument("Error while
entering expression");

    }

}

else if (k == ' ')

    Num = 0;

else

    throw std::invalid_argument("Error while entering
expression");

return Num;

}

if (k < '0' || k > '9')

```

```

        throw std::invalid_argument("Error while entering expression");

    Num += ((base)k - '0');

    std::cout << k;

    while (1)
    {
        if (!Stream.get(k))

            throw std::invalid_argument("Error while entering
expression");

        if (k >= '0' && k <= '9')
        {
            Num *= 10;

            Num += ((base)k - '0');

            std::cout << k;

        }

        else if (k == ' ')

            return Num;

        else if (k == ',')
        {

            std::cout << ',';

            int I = 0;

            if (!Stream.get(k))

                throw std::invalid_argument("Error while entering
expression");

```

```

        if (k < '0' || k > '9')

            throw std::invalid_argument("Error while entering
expression");

        Num += 0.1 * ((base)k - '0');

        I++;

        std::cout << k;

        while (1)

        {

            if (!Stream.get(k))

                throw std::invalid_argument("Error while
entering expression");

            if (k >= '0' && k <= '9')

            {

                base deg = 0.1;

                for (int i = 0; i < I; i++)

                    deg /= 10;

                Num += deg * ((base)k - '0');

                I++;

                std::cout << k;

            }

            else if (k == ' ')

                return Num;

            else

```

```

        throw std::invalid_argument("Error while
entering expression");
    }
}
else
    throw std::invalid_argument("Error while entering
expression");
}
}

void ReadNumberToH_List(std::shared_ptr<h_list<base>> tmp, char k,
std::stringstream& Stream) {
    VarNum<base> Num;
    Num.IsVar = false;
    Num.IsOp = false;
    Num.Num = ReadNumber(k, Stream);
    tmp->value = Num;
    std::cout << ' ';
}

void ReadVar(std::shared_ptr<h_list<base>> tmp, char v, std::stringstream&
Stream) {
    VarNum<base> V;
    V.IsVar = true;
    V.IsOp = false;
    std::cout << v;

```

```

V.Var = v;

tmp->value = V;

if (!Stream.get(v))

    throw std::invalid_argument("Error while entering expression");

if (v != ' ')

    throw std::invalid_argument("Error while entering expression");

std::cout << ' ';

}

void ReadOper(std::shared_ptr<h_list<base>> tmp, char v, std::stringstream&
Stream) {

    VarNum<base> Op;

    Op.IsVar = false;

    Op.IsOp = true;

    std::cout << v;

    Op.Var = v;

    tmp->value = Op;

    if (!Stream.get(v))

        throw std::invalid_argument("Error while entering expression");

    if (v != ' ')

        throw std::invalid_argument("Error while entering expression");

    std::cout << ' ';

}

```

```

void ReadExprRec(std::shared_ptr<h_list<base>> tmp,
std::shared_ptr<h_list<base>> &save, std::stringstream &Stream) {

    int IsO = 0;

    std::cout << '(';

    char c = 0;

    while (Stream.get(c))

    {

        if (IsO < 2)

        {

            if (c >= '0' && c <= '9')

            {

                ReadNumberToH_List(tmp, c, Stream);

                tmp->AddNext();

                tmp = tmp->next;

                IsO++;

            }

            else if (c >= 'a' && c <= 'z')

            {

                ReadVar(tmp, c, Stream);

                tmp->AddNext();

                tmp = tmp->next;

                IsO++;

            }

        }

    }

}

```



```

    }

    else if (c == '(')
    {
        if (!Stream.get(c))
            throw std::invalid_argument("Error while
entering expression");

        if (c != ' ')
            throw std::invalid_argument("Error while
entering expression");

        tmp->value = std::make_shared<h_list<base>>>();

        ReadExprRec(std::get<std::shared_ptr<h_list<base>>>(tmp->value),
            std::get<std::shared_ptr<h_list<base>>>(tmp-
>value), Stream);

        IsO++;

        tmp->AddNext();

        tmp = tmp->next;
    }

    else if (c == ')') && IsO == 1)
    {
        if (!Stream.get(c))
            throw std::invalid_argument("Error while
entering expression");

        if (c != ' ')

```

```

        throw std::invalid_argument("Error while
entering expression");

        IsO++;

        std::cout << "\b\b";

        std::cout << ')';

        std::cout << ' ';

        return;
    }

    else

        throw std::invalid_argument("Error while entering
expression");

    }

    else if (IsO == 2)

    {

        if ((c == '+' || c == '-' || c == '*' || c == '/'))

        {

            ReadOper(tmp, c, Stream);

            IsO = 1;

            H_ListToValueOfRoot(tmp, save);

            tmp->AddNext();

            tmp = tmp->next;

        }

    else

```

```

        throw std::invalid_argument("Error while entering
expression");

    }

}

throw std::runtime_error("Error while reading from stream");

}

bool ReadVarValue(std::stringstream& Stream) {

    std::cout << '(';

    char k = 0;

    if (!Stream.get(k))

        throw std::invalid_argument("Error while entering expression");

    if (k != '(')

        throw std::invalid_argument("Error while entering expression");

    if (!Stream.get(k))

        throw std::invalid_argument("Error while entering expression");

    if (k != ' ')

        throw std::invalid_argument("Error while entering expression");

    if (!Stream.get(k))

        throw std::invalid_argument("Error while entering expression");

    if (k == ')')

    {

        std::cout << ')';

        return 0;
    }

```

```

    }

    else if (k >= 'a' && k <= 'z')
    {
        std::cout << k << ' ';

        char c = 0;

        if (!Stream.get(c))

            throw std::invalid_argument("Error while entering
expression");

        if (c != ' ')

            throw std::invalid_argument("Error while entering
expression");

        if (!Stream.get(c))

            throw std::invalid_argument("Error while entering
expression");

        VarValue[k] = new (base);

        *VarValue[k] = ReadNumber(c, Stream);

        std::cout << ')';

        if (!Stream.get(c))

            throw std::invalid_argument("Error while entering
expression");

        if (c != ')')

            throw std::invalid_argument("Error while entering
expression");

        if (!Stream.get(c))

```

```

        throw std::invalid_argument("Error while entering
expression");

        if (c == ' ')

            return 0;

        else if (c == ',')

        {

            std::cout << ',';

            return ReadVarValue(Stream);

        }

    }

    else

        throw std::invalid_argument("Error while entering expression");

    return 0;

}

base CalcExprRec(std::shared_ptr<h_list<base>> tmp) {

    base O1, O2, Res = 0;

    if (tmp->next == nullptr &&
std::holds_alternative<std::shared_ptr<h_list<base>>>(tmp->value) &&
std::get<std::shared_ptr<h_list<base>>>(tmp->value) == nullptr)

        throw std::invalid_argument("Empty hierarchical list");

    if (tmp->next == nullptr || (tmp->next->next == nullptr &&
std::holds_alternative<std::shared_ptr<h_list<base>>>(tmp->next->value) &&
std::get<std::shared_ptr<h_list<base>>>(tmp->next->value) == nullptr))

```

```

        if (std::holds_alternative<std::shared_ptr<h_list<base>>>(tmp-
>value))

            Res =
CalcExprRec(std::get<std::shared_ptr<h_list<base>>>(tmp->value));

        else

        {

            VarNum N = std::get<VarNum<base>>(tmp->value);

            if (N.IsVar == 0)

                Res = N.Num;

            else

            {

                if (VarValue[N.Var] == 0)

                    throw std::invalid_argument("Uninitialized
variable");

                Res = *VarValue[N.Var];

            }

        }

    else

    {

        if (std::holds_alternative<std::shared_ptr<h_list<base>>>(tmp-
>value))

            O1 =
CalcExprRec(std::get<std::shared_ptr<h_list<base>>>(tmp->value));

        else

```

```

{
    VarNum N = std::get<VarNum<base>>(tmp->value);

    if (N.IsVar == 0)

        O1 = N.Num;

    else

    {

        if (VarValue[N.Var] == 0)

            throw std::invalid_argument("Uninitialized
variable");

        O1 = *VarValue[N.Var];

    }

    tmp = tmp->next;

    if (std::holds_alternative<std::shared_ptr<h_list<base>>>(tmp-
>value))

        O2 =
CalcExprRec(std::get<std::shared_ptr<h_list<base>>>(tmp->value));

    else

    {

        VarNum N = std::get<VarNum<base>>(tmp->value);

        if (N.IsVar == 0)

            O2 = N.Num;

        else

        {

```

```

        if (VarValue[N.Var] == 0)

            throw std::invalid_argument("Uninitialized

variable");

        O2 = *VarValue[N.Var];

    }

}

tmp = tmp->next;

VarNum N = std::get<VarNum<base>>(tmp->value);

switch (N.Var)

{

case '+':

    Res = O1 + O2;

    break;

case '-':

    Res = O1 - O2;

    break;

case '*':

    Res = O1 * O2;

    break;

case '/':

    Res = O1 / O2;

    break;

}

```



```

    }

    return Res;

}

public:

    calc(h_list<base> L) {

        H_List = std::make_shared<h_list<base>>(L);

    }

    void ReadExpr(std::stringstream& Stream) {

        char c = 0;

        if (!Stream.get(c))

            throw std::invalid_argument("Error while entering expression");

        if (c != '(')

            throw std::invalid_argument("Error while entering expression");

        if (!Stream.get(c))

            throw std::invalid_argument("Error while entering expression");

        if (c != ' ')

            throw std::invalid_argument("Error while entering expression");

        ReadExprRec(H_List, H_List, Stream);

        std::cout << ' ';

        ReadVarValue(Stream);

    }

    base CalcExpr(void) {

        return CalcExprRec(H_List);
    }

```

```
    }  
};
```

```
#endif // __CALC
```

Название файла: main.cpp

```
#include <stdexcept>  
#include <string>  
#include "../Lab_2/calc.h"  
#include <fstream>  
int main() {  
    try  
    {  
        double current_results[] = { 3, 2, 5, 1, 15, 0, 0 };  
        h_list<double> L;  
        calk<double> C(L);  
        double Res;  
        std::string Str;  
        /* Test 1  
        std::ifstream InTest1("../tests/test_in/in1.txt");  
        if (InTest1.is_open())  
        {  
            if (!std::getline(InTest1, Str))  
                throw std::runtime_error("Error while reading from stream");  
        }  
        std::stringstream Stream(Str);  
        C.ReadExpr(Stream);  
        Res = C.CalcExpr();
```

```

std::cout << '\n' << Res << '\n';
if (Res == current_results[0])
    std::cout << '\n' << "test passed" << '\n';
else
    std::cout << '\n' << "test failed" << '\n';
/**/
/* Test 2
std::ifstream InTest1("../tests/test_in/in2.txt");
if (InTest1.is_open())
{
    if (!std::getline(InTest1, Str))
        throw std::runtime_error("Error while reading from stream");
}
std::stringstream Stream(Str);
C.ReadExpr(Stream);
Res = C.CalcExpr();
std::cout << '\n' << Res << '\n';
if (Res == current_results[1])
    std::cout << '\n' << "test passed" << '\n';
else
    std::cout << '\n' << "test failed" << '\n';
/**/
/* Test 3
std::ifstream InTest1("../tests/test_in/in3.txt");
if (InTest1.is_open())
{
    if (!std::getline(InTest1, Str))
        throw std::runtime_error("Error while reading from stream");
}

```

```

std::stringstream Stream(Str);
C.ReadExpr(Stream);
Res = C.CalcExpr();
std::cout << '\n' << Res << '\n';
if (Res == current_results[2])
    std::cout << '\n' << "test passed" << '\n';
else
    std::cout << '\n' << "test failed" << '\n';
//*/
/* Test 4
std::ifstream InTest1("../tests/test_in/in4.txt");
if (InTest1.is_open())
{
    if (!std::getline(InTest1, Str))
        throw std::runtime_error("Error while reading from stream");
}
std::stringstream Stream(Str);
C.ReadExpr(Stream);
Res = C.CalcExpr();
std::cout << '\n' << Res << '\n';
if (Res == current_results[3])
    std::cout << '\n' << "test passed" << '\n';
else
    std::cout << '\n' << "test failed" << '\n';
//*/
/* Test 5
std::ifstream InTest1("../tests/test_in/in5.txt");
if (InTest1.is_open())
{

```

```

        if (!std::getline(InTest1, Str))
            throw std::runtime_error("Error while reading from stream");
    }
    std::stringstream Stream(Str);
    C.ReadExpr(Stream);
    Res = C.CalcExpr();
    std::cout << '\n' << Res << '\n';
    if (Res == current_results[4])
        std::cout << '\n' << "test passed" << '\n';
    else
        std::cout << '\n' << "test failed" << '\n';
    /**/
    /* Test 6
    std::ifstream InTest1("../tests/test_in/in6.txt");
    if (InTest1.is_open())
    {
        if (!std::getline(InTest1, Str))
            throw std::runtime_error("Error while reading from stream");
    }
    std::stringstream Stream(Str);
    C.ReadExpr(Stream);
    Res = C.CalcExpr();
    std::cout << '\n' << Res << '\n';
    if (Res == current_results[6])
        std::cout << '\n' << "test passed" << '\n';
    else
        std::cout << '\n' << "test failed" << '\n';
    /**/
    /* Test 7

```

```

std::ifstream InTest1("../tests/test_in/in7.txt");
if (InTest1.is_open())
{
    if (!std::getline(InTest1, Str))
        throw std::runtime_error("Error while reading from stream");
}
std::stringstream Stream(Str);
C.ReadExpr(Stream);
Res = C.CalcExpr();
std::cout << '\n' << Res << '\n';
if (Res == current_results[6])
    std::cout << '\n' << "test passed" << '\n';
else
    std::cout << '\n' << "test failed" << '\n';
//*/
system("pause");
}
catch (const std::exception& Error)
{
    std::cout << '\n' << Error.what();
    std::cout << '\n' << "test failed" << '\n';
}
return 0;
}

```

Название файла: Makefile

lab2: ./src/main.cpp

g++ -std=c++17 ./Src/main.cpp -o lab2

tests: ./lab2\_tests/main.cpp

```
g++ -std=c++17 ./lab2_tests/main.cpp -o tests
```

## **ПРИЛОЖЕНИЕ Б**

### **ТЕСТИРОВАНИЕ**

Таблица Б.1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарии
1	( 1 2 + ) ( )	(1 2 +) () 3  test passed	Сложение двух чисел

2	$(a)(a^2)$	$(a)(a^2)$ 2  test passed	Инициализация переменной
3	$(ab + 2 -)(a^3), (b^4)$	$(ab + 2 -)(a^3), (b^4)$ 5  test passed	Подсчет выражения и с переменными, и с числом
4	$((23 +)(ab +) -)(a^0), (b^4)$	$((23 +)(ab +) -)(a^0), (b^4)$ 1  test passed	Подсчет выражения со вложенными скобками
5	$(a(3,29(2v -) +) *)(v^0,29), (a^3)$	$(a(3,29(2v -) +) *)(v^0,29), (a^3)$ 15  test passed	Подсчет выражения с двойной вложенностью
6	$(ab -)(a^3)$	$(ab -)(a^3)$ Uninitialized variable test failed	Выражение с неинициализированной переменной
7	$(23 +))$	$(23 +$ Error while entering expression test failed	Неправильная запись выражения