

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

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

Выводы.

Стало известно о иерархическом списке и его использовании в практике.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <stdexcept>
#include "Calc.h"

int main()
{
    try
    {
        h_list<double> L;
        calk<double> C(L);
        double Res;
        C.ReadExpr();
        Res = C.CalcExpr();
        std::cout << '\n' << Res;
        system("pause");
    }
    catch (const std::exception& Error)
    {
        std::cout << 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_lis
```

```
t>());
```

```
}
```

```
};
```

```
#endif // __H_LIST
```

Название файла: calc.h

```
#ifndef __CALC
```

```
#define __CALC
```

```
#include "h_list.h"
```

```
#include <stdexcept>
```

```
#include <conio.h>
```

```
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) {
```

```
        base Num = 0;
```

```
        if (k == '0')
```

```
        {
```

```
            std::cout << '0';
```

```
            k = _getch();
```



```

if (k == ',')
{
    std::cout << ',';
    int I = 0;
    while (k != 13)
    {
        if (I == 0)
            while (k < '0' || k > '9')
                k = _getch();
        else
            k = _getch();
        if (k >= '0' && k <= '9')
        {
            std::cout << k;
            base deg = 0.1;
            for (int i = 0; i < I; i++)
                deg /= 10;
            Num += deg * ((base)k - '0');
            I++;
        }
    }
}
else if (k == '13')
    Num = 0;

return Num;
}
while (1)
{
    if (k >= '0' && k <= '9')

```

```

    {
        std::cout << k;
        Num *= 10;
        Num += ((base)k - '0');
        break;
    }
    k = _getch();
}
while (k != 13)
{
    k = _getch();
    if (k >= '0' && k <= '9')
    {
        std::cout << k;
        Num *= 10;
        Num += ((base)k - '0');
    }
    else if (k == ',')
    {
        std::cout << ',';
        int I = 0;
        while (k != 13)
        {
            if (I == 0)
                while (k < '0' || k > '9')
                    k = _getch();
            else
                k = _getch();
            if (k >= '0' && k <= '9')
            {

```

```

        std::cout << k;
        base deg = 0.1;
        for (int i = 0; i < I; i++)
            deg /= 10;
        Num += deg * ((base)k - '0');
        I++;
    }
}

}

}

return Num;
}

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

void ReadVar(std::shared_ptr<h_list<base>> tmp, char v) {
    VarNum<base> V;
    V.IsVar = true;
    V.IsOp = false;
    std::cout << v;
    V.Var = v;
    tmp->value = V;
    std::cout << ' ';
}

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

```

```

    VarNum<base> Op;
    Op.IsVar = false;
    Op.IsOp = true;
    std::cout << v;
    Op.Var = v;
    tmp->value = Op;
    std::cout << ' ';
}

void ReadExprRec(std::shared_ptr<h_list<base>> tmp,
std::shared_ptr<h_list<base>> &save) {
    int IsO = 0;
    std::cout << '(';
    while (1)
    {
        int c = _getch();
        if (IsO < 2)
            while (1)
            {
                if (c >= '0' && c <= '9')
                {
                    ReadNumberToH_List(tmp, c);
                    tmp->AddNext();
                    tmp = tmp->next;
                    IsO++;
                    break;
                }
                else if (c >= 'a' && c <= 'z')
                {
                    ReadVar(tmp, c);
                    tmp->AddNext();

```

```

        tmp = tmp->next;
        IsO++;
        break;
    }
    else if (c == '(')
    {
        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));

        IsO++;
        tmp->AddNext();
        tmp = tmp->next;
        break;
    }
    else if (c == ')')
    {
        if (IsO == 1)
        {
            IsO++;
            std::cout << "\b \b";
            std::cout << ')';
            std::cout << ' ';
            return;
        }
    }
    else
        c = _getch();
}

```

```

        //int c = _getch();
        if (IsO == 2)
            while (1)
            {
                if ((c == '+' || c == '-' || c == '*' || c == '/'))
                {
                    ReadOper(tmp, c);
                    IsO = 1;
                    H_ListToValueOfRoot(tmp, save);
                    tmp->AddNext();
                    tmp = tmp->next;
                    break;
                }
                else
                    c = _getch();
            }
    }
}

bool ReadVarValue() {
    std::cout << '(';
    char k = _getch();
    while (k < 'a' || k > 'z')
        k = _getch();
    std::cout << k << ' ';
    char c = _getch();
    while (c < '0' || c > '9')
        c = _getch();
    VarValue[k] = ReadNumber(c);
    std::cout << ')';
    k = _getch();
}

```

```

while (1)
{
    if (k == 13)
        return 0;
    else if (k == ',')
        return ReadVarValue();
    k = _getch();
}
}

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

```

```
void ReadExpr(void) {  
    ReadExprRec(H_List, H_List);  
    std::cout << '\n';  
    ReadVarValue();  
}  
base CalcExpr(void) {  
    return CalcExprRec(H_List);  
}  
};  
  
#endif // __CALC
```