

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

Шуняев А.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием иерархического списка. Реализовать иерархический список на языке программирования C++.

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.

№19

арифметическое, проверка синтаксической корректности и деления на 0 (простая), постфиксная форма

Описание алгоритма работы программы.

На вход программы подаются строки неограниченной длины. Данные строки записываются в список строк. После этого вызывается рекурсивный метод, который создает иерархический список. Далее этот список обрабатывается. Сама обработка заключается в проверки элементов списка. Сначала проверяется не пустой ли список, затем проверяется заканчивается ли список знаком операции, после этого проверяется каждый элемент списка. Если встречается узел списка, то обработка вызывается рекурсивно для списка, на который указывает узел. Эта обработка возвращает булево значение. Данное булево значение определяет ответ, который будет записан в файл вывода.

Формат входных и выходных данных

Входные данные представлены в виде строк. В них входят константы и переменный, которые являются атомами списка. Операции представлены в постфиксной форме (($\langle \text{аргументы} \rangle \langle \text{операция} \rangle$)). Аргументов может быть 1,2 или более. Пример: $(a(aa+)^*)$ / Они считываются из файла input.txt. Выходными данными являются те же строки, плюс строки-индикаторы “ – CORRECT” и “ – INCORRECT”, которые указывают является ли строка синтаксически корректной или нет. Выходные данные записываются в файлы output.txt.

Описание основных структур данных и функций (кратко).

- Структура Node – элемент иерархического списка
- Метод InputData – считывает входные данные из файла
- Статический метод CreateHList – создает иерархический список
- Метод IsCorrect – проверяет список на корректность

Тестирование

Входные данные	Выходные данные	Результат теста
(aaaaaaa)	(aaaaaaa) - INCORRECT!	Program anwer: (aaaaaaa)-INCORRECT! True anwer: (aaaaaaa) - INCORRECT! Test #1 - True!
(23d+)	(23d+) - CORRECT!	Program anwer: (23d+) - CORRECT! True anwer: (23d+) - CORRECT! Test #2 - True!
(230d(ab+)/)	(230d(ab+)/) - INCORRECT!	Program anwer: (230d(ab+)/) -INCORRECT! True anwer: (230d(ab+)/) - INCORRECT! Test #3 - True!
(23d/)	(23d/) - CORRECT!	Program anwer: (23d/) - CORRECT! True anwer: (23d/) - CORRECT!

был сделан вывод, что иерархические списки очень неудобная система хранения данных, т.к. неудобно обращаться к вложенным спискам и следить, чтобы всё было просмотрено.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ.

Файл mail.cpp:

```
#include "Algorithm.h"
#include <crtdbg.h>
```

```
int main() {
    Algorithm::StartDataProcessing();

    _CrtDumpMemoryLeaks();
    return 0;
}
```

Файл Algorithm.h:

```
#pragma once
#include <iostream>
#include <fstream>
#include <list>
#include "HierarchicalList.h"
```

```
class Algorithm
{
public:
    static void StartDataProcessing();

    bool IsCorrect(std::shared_ptr<HierarchicalList> h_list);
    std::list<std::string> InputData();

};
```

Файл Algorithm.cpp:

```
#include "Algorithm.h"

void Algorithm::StartDataProcessing()
{
    Algorithm algorithm;

    std::list<std::string> input_data = algorithm.InputData();

    std::ofstream output("../Tests/output.txt");

    while (input_data.size() != 0) {

        std::size_t t = input_data.front().find("Testing the");
        if (t == 0) {
```

```

        output << '\n';
        output << input_data.front() << "\n\n";
        input_data.pop_front();
        continue;
    }
    else {
        if (input_data.front()[0] == '(') {
            int position = 1;
            std::shared_ptr<HierarchicalList> h_list =
HierarchicalList::CreateHList(input_data.front(), position);

            if (output.is_open()) {
                if (algorithm.IsCorrect(h_list)) {
                    output << input_data.front() << " - CORRECT!" << '\n';
                }
                else {
                    output << input_data.front() << " - INCORRECT!" << '\n';
                }
                input_data.pop_front();
            }
            else {
                std::cout << "Can't open output file!" << std::endl;
            }
        }
        else {
            output << input_data.front() << " - INCORRECT!" << '\n';
            input_data.pop_front();
        }
    }

}

output.close();
}

bool Algorithm::IsCorrect(std::shared_ptr<HierarchicalList> h_list)
{
    bool is_correct = false;
    bool is_div = false;
    std::shared_ptr<Node> temp1 = h_list->Back();
    if (h_list->IsAtom(temp1) && (std::get<char>(temp1->value) == '+' || std::get<char>(temp1->value) == '-'
                                                                    || std::get<char>(temp1->value) == '*' ||
std::get<char>(temp1->value) == '/'))
    {
        if (std::get<char>(temp1->value) == '/') {
            is_div = true;
        }
    }
}

```

```

        is_correct = true;
    }

    std::shared_ptr<Node> temp = h_list->Front();

    for (int i = 0; i < h_list->Length(); i++) {
        if (!is_correct) {
            break;
        }

        if (h_list->IsAtom(temp)) {

            char temp_char = std::get<char>(temp->value);

            if ((temp_char >= 'a' && temp_char <= 'z') || (temp_char >= '0' && temp_char <= '9'))
            {
                if (temp_char == '0' && is_div) {
                    is_correct = false;
                }
                temp = temp->next;
            }
            else if (temp_char == '+' || temp_char == '-' || temp_char == '*' || temp_char == '/') {
                if (temp->next != nullptr )
                {
                    is_correct = false;
                }
                else {
                    is_correct = true;
                }
            }
            else {
                is_correct = false;
            }
        }
        else {
            is_correct = this->IsCorrect(std::get<std::shared_ptr<HierarchicalList>>(temp->value));
            temp = temp->next;
        }
    }
    return is_correct;
}

std::list<std::string> Algorithm::InputData()
{
    std::ifstream input("../Tests/input.txt");
    std::list<std::string> list;
    std::string str;

```



```

        if (input.is_open()) {

            while (std::getline(input, str)) {
                list.push_back(str);
            }
        }
        else {
            std::cout << "Can't open input file!" << std::endl;
        }

        input.close();

        return list;
    }

```

Файл HierarchicalList.h:

```

#pragma once
#include <variant>
#include <string>
#include <memory>

class HierarchicalList;

struct Node
{
    std::shared_ptr<Node> next = nullptr;
    std::variant<std::shared_ptr<HierarchicalList>, char> value;
};

class HierarchicalList
{
public:

    ~HierarchicalList();

    void PushBack(std::shared_ptr<Node>);
    bool IsAtom(std::shared_ptr<Node>);

    static std::shared_ptr<HierarchicalList> CreateHList(std::string& str, int& position);

    std::shared_ptr<Node> Front();
    std::shared_ptr<Node> Back();
    int Length();

private:

```

```

        int length_ = 0;
        std::shared_ptr<Node> head_ = nullptr;
        std::shared_ptr<Node> end_ = nullptr;
    };

```

Файл HierarchicalList.cpp:

```
#include "HierarchicalList.h"
```

```
HierarchicalList::~HierarchicalList()
```

```
{
}
```

```
void HierarchicalList::PushBack(std::shared_ptr<Node> element)
```

```
{
    if (this->head_ == nullptr) {
        this->head_ = element;
        this->end_ = this->head_;
    }
    else {
        this->end_->next = element;
        this->end_ = element;
    }

    this->length_++;
}
```

```
bool HierarchicalList::IsAtom(std::shared_ptr<Node> element)
```

```
{
    if (std::holds_alternative<std::shared_ptr<HierarchicalList>>(element->value)) {
        return false;
    }
    else {
        return true;
    }
}
```

```
std::shared_ptr<Node> HierarchicalList::Front()
```

```
{
    return this->head_;
}
```

```
std::shared_ptr<Node> HierarchicalList::Back()
```

```
{
    return this->end_;
}
```

```
int HierarchicalList::Length()
```

```

{
    return this->length_;
}

std::shared_ptr<HierarchicalList> HierarchicalList::CreateHList(std::string& str, int& position)
{
    std::shared_ptr<HierarchicalList> h_list = std::make_shared<HierarchicalList>();
    for (position; position < str.length(); position++) {
        if (str[position] == '(') {
            std::shared_ptr<Node> node = std::make_shared<Node>();
            std::shared_ptr<HierarchicalList> temp = HierarchicalList::CreateHList(str, ++position);
            node->value = temp;
            h_list->PushBack(node);
        }
        else if (str[position] == ')') {
            break;
        }
        else {
            std::shared_ptr<Node> atom = std::make_shared<Node>();
            atom->value = str[position];
            h_list->PushBack(atom);
        }
    }
    return h_list;
}

```