

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

Атаманов С.Д.

Преподаватель

Филатов Ар. Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием иерархического списка. Реализовать иерархический список, используя язык программирования C++.

Задание.

Вариант 10.

Подсчитать число различных атомов в иерархическом списке; сформировать из них линейный список.

Выполнение работы.

Описание алгоритма работы.

Класс Node описывает узлы иерархического списка. Класс содержит два публичных поля: `nex` — которое указывает на следующий элемент поля, и `value` — которое хранит в себе значение узла. Хранение осуществляется с помощью универсального типа `std::variant`.

Программа начинается с инициализации переменной строкового типа `stringList`. Далее происходит считывание строки из `stdin`, считанная строка записывается в переменную `stringList`. С помощью функции `getConvinientLine()` строка очищается от пробелов. В данной функции создается новая пустая строка, в которую записываются все символы, не равные пробелу. Функция возвращает очищенную строку. С помощью функции `isCorrect()` происходит проверка строки на валидность. Элементы строки поочередно добавляются в стек, если встречен символ `(`. Если встречен символ `)`, то элемент удаляется из стека. Функция возвращает `true` в случае, если стек пуст и `false`, если стек непуст, или была произведена попытка извлечь элемент из пустого стека.

Далее инициализируется итератор строки, для считывания введенного иерархического списка. Создается объект класса Node — `list`. С помощью рекурсивной функции `getList()` происходит преобразование строки в список.

Функция инициализирует объект класса Node, который будет хранить временное значение списка. Далее с помощью цикла `while` происходит проход по строке с помощью итератора. Если значение итератора `==` открывающей скобке, то в значение узла записывается новый элемент списка(опускаемся на

уровень ниже), который вычисляется рекурсивно. Если встречено значение != закрывающей скобке, то в значение узла записывается значение, хранящееся в итераторе, а значение next вычисляется рекурсивно. Если встречено значение == закрывающей скобке, то происходит выход из функции; условие выхода определяется с помощью условных операторов.

Далее, с помощью рекурсивной функции getDiffAtomList() происходит проход по реализованному списку и считывание значений их атомов. С помощью вектора реализован поиск различных элементов. Если элемент встретился первый раз, то он добавляется в вектор, иначе функция просто продолжает свою работу. Далее различные атомы добавляются в линейный список, который реализован на базе класса Node, но элементами узла могут быть только атомы.

Для более удобной работы с памятью, были добавлены умные указатели.

Для универсальности программы все функции реализованы с помощью шаблонов.

В конце работы сформированный список выводится на экран.

Также был написан make файл, который позволяет удобно собирать программу и проводить тестирование.

Разработанный программный код смотри в приложении А.

Формат входных и выходных данных .

Программа получает строку, содержащую строковое представление списка. Между ними может быть любое кол-во пробелов. Элементами могут быть любые стандартные типы C++.

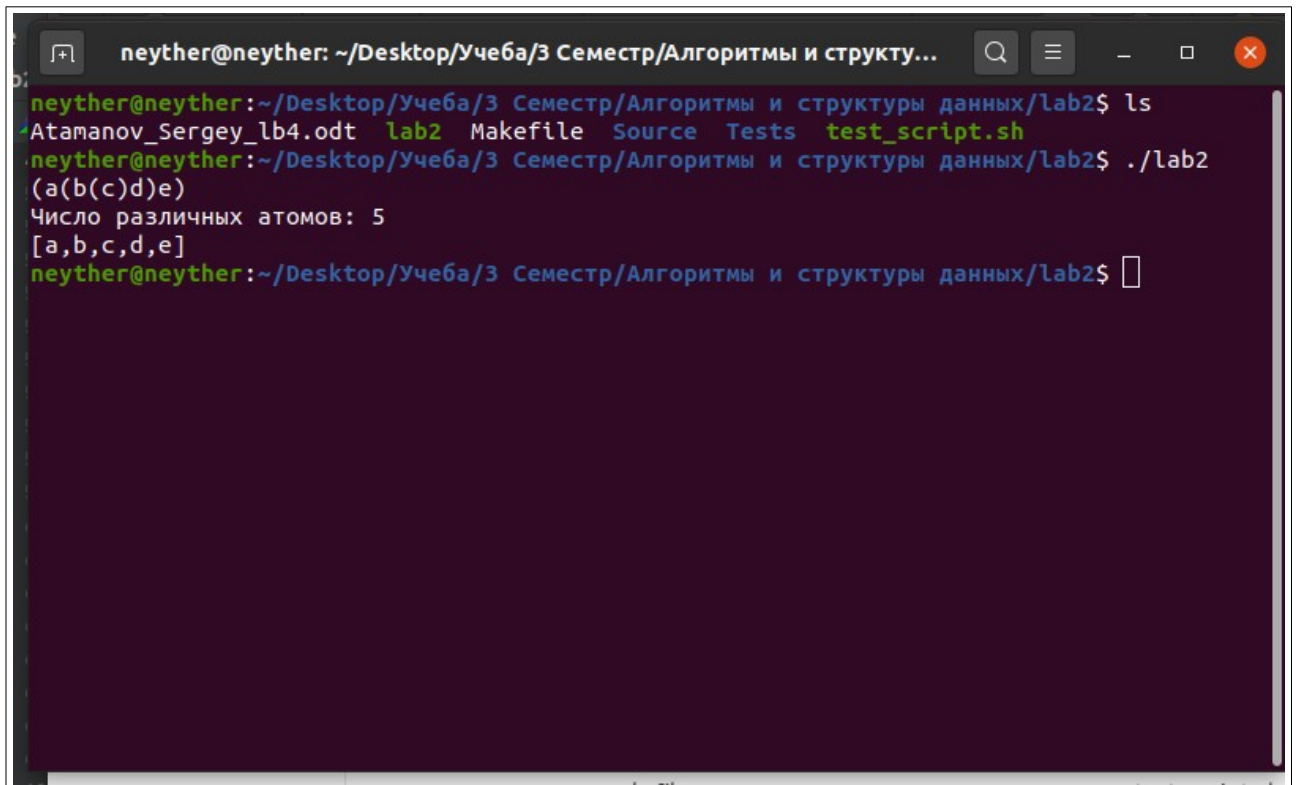
Считывание происходит из файла.

Программа выводит промежуточные и окончательные результаты в stdout.

Тестирование.

Для тестирования был написан bash-скрипт, который считывает тестовые данные из текстовых документов папки tests и передает их программе. В самой программе происходят необходимые операции с иерархическими списками.

Данные о результатах тестирования выводятся в stdout, которые записываются в файлы папки Output, которые затем сравниваются с правильными результатами, которые хранятся в папке Test_result.



```
neyther@neyther: ~/Desktop/Учеба/3 Семестр/Алгоритмы и структу...
neyther@neyther:~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/lab2$ ls
Atamanov_Sergey_lb4.odt lab2 Makefile Source Tests test_script.sh
neyther@neyther:~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/lab2$ ./lab2
(a(b(c)d)e)
Число различных атомов: 5
[a,b,c,d,e]
neyther@neyther:~/Desktop/Учеба/3 Семестр/Алгоритмы и структуры данных/lab2$
```

Рисунок 1 — Пример работы программы

Результаты тестирования смотри в приложении Б.

Выводы.

В ходе выполнения работы были получены навыки работы с иерархическими списками.

Была разработана программа на языке C++, которая получает на вход строковое представление списка, затем формирует из его атомов линейный список и выводит его на экран.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

Название файла: lab2.cpp

```
#include <iostream>
#include <string>
#include <memory>
#include <vector>
#include <stack>
#include <variant>
#include <algorithm>

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

    std::shared_ptr<Node<base>> getList(typename base::const_iterator &begin,
std::shared_ptr<Node<base>> &head) {
        std::shared_ptr<Node<base>> temp;
        base listValue;
        while (*begin != '\0') {
            if (*begin == '(') {
                if (temp == nullptr) {
                    temp = std::make_shared<Node<base>>();
                    begin++;
                    temp->value = getList(begin, head);
                } else {
                    begin++;
                    temp->value = getList(begin, head);
                }
            } else if (*begin != ')') {
                if (temp == nullptr) {
                    temp = std::make_shared<Node<base>>();
                    listValue = *begin;
                    temp->value = listValue;
                    begin++;
                    temp->next = getList(begin, head);
                } else {
                    listValue = *begin;
                    temp->value = listValue;
                    begin++;
                    temp->next = getList(begin, head);
                }
            } else {
                if (temp == nullptr) {
                    return temp;
                } else if (temp->next == nullptr && std::holds_alternative<base>(temp->value)) {
                    return temp;
                } else if (temp->next != nullptr && std::holds_alternative<base>(temp->value)) {
                    return temp;
                } else if (temp->next != nullptr && !std::holds_alternative<base>(temp->value)) {
                    return temp;
                } else {
                    if (*(++begin) == '\0')
                        return temp;
                    else
                        temp->next = getList(begin, head);
                }
            }
        }
    }
};
```

```

    }
    }
}

void getDiffAtomList(std::shared_ptr<Node<base>> list, std::vector<base> &values,
std::shared_ptr<Node<base>> &output, std::shared_ptr<Node<base>> &head) {
    while (list != nullptr) {
        if (std::holds_alternative<base>(list->value)) {
            if (std::find(values.begin(), values.end(), std::get<base>(list->value)) ==
values.end()) {
                values.push_back(std::get<base>(list->value));
                if (head == nullptr) {
                    head = std::make_shared<Node<base>>();
                    head->value = std::get<base>(list->value);
                    output = head;
                } else {
                    output->next = std::make_shared<Node<base>>();
                    output = output->next;
                    output->value = std::get<base>(list->value);
                }
            }
            } else {
                getDiffAtomList(std::get<std::shared_ptr<Node<base>>>(list->value), values,
output, head);
            }
            list = list->next;
        }
    }
};

std::string getConvenientLine(std::string::iterator iter){
    std::string output;
    output.clear();
    while(*iter != '\0'){
        if(*iter != ' '){
            output.push_back(*iter);
            iter++;
        }
    }
    return output;
}

bool isCorrect(std::string::const_iterator& begin){
    std::stack<char> Stack;
    if (*begin != '('){
        return false;
    }
    for (;*begin != '\0';begin++){
        if (*begin == '('){
            Stack.push(*begin);
        }
        if (*begin == ' '){
            if (Stack.empty()){
                return false;
            }
            Stack.pop();
        }
    }
    return Stack.empty();
}

```

```

int main(){
    std::string stringList;
    std::getline(std::cin, stringList);
    auto iterCheck = stringList.begin();
    stringList = getConvenientLine(iterCheck);
    auto iterBeg = stringList.cbegin();
    if(!isCorrect(iterBeg)){
        std::cout << "Wrong stringList format\n";
        exit(EXIT_FAILURE);
    }
    iterBeg = stringList.cbegin();
    std::shared_ptr<Node<std::string>> list;
    list = list->getList(iterBeg, list);
    std::vector<std::string> atoms;
    std::shared_ptr<Node<std::string>> atomList, atomHead;
    list->getDiffAtomList(std::get<std::shared_ptr<Node<std::string>>>(list->value), atoms,
atomList, atomHead);
    atomList = atomHead;
    std::cout << "Число различных атомов: " << atoms.size() << "\n";
    std::cout << '[';
    while(atomList != nullptr){
        if(atomList -> next == nullptr){
            std::cout << std::get<std::string>(atomList->value);
            break;
        }
        std::cout << std::get<std::string>(atomList->value) << ',';
        atomList = atomList->next;
    }
    std::cout << ']';
    return 0;
}

```

Название файла: Makefile

```

lab2: Source/lab2.cpp
    g++ -std=c++17 Source/lab2.cpp -o lab2
run_tests:
    ./test_script.sh
clean:
    rm lab2

```

ПРИЛОЖЕНИЕ Б.
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ.

Входные данные	Выходные данные	Комментарий
(a (b (c d)g) a)	Число различных атомов: 5 [a,b,c,d,g]	Обычный список с пробелами и одним повторяющимся элементом
((()))	Число различных атомов: 0 []	Список без атомов
(a(aaa(a)aaa)a)	Число различных атомов: 1 [a]	Список, атомами которого является одно повторяющееся значение
(abcdefgh)	Число различных атомов: 8 [a,b,c,d,e,f,g,h]	Различные атомы, расположенные на 1 уровне списка
(a(b(c(a(b(c(d)))a)b)c)	Число различных атомов: 4 [a,b,c,d]	Список с повторяющимися атомами
(aaaaaaaaaaa(b bbbbbb(cdef)gzl oiu)mnb)	Число различных атомов: 14 [a,b,c,d,e,f,g,z,l,o,i,u,m,n]	Список с повторяющимися атомами
	Result: Wrong stringList format	Пустой файл
(abcd	Result: Wrong stringList format	Неправильный формат списка
((a))	Result: Wrong stringList format	Неправильный формат списка