

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

Цаплин И.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

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

Задание.

Вариант 9.

Подсчитать число атомов в иерархическом списке. Сформировать линейный список атомов, соответствующий порядку подсчёта.

Описание алгоритма работы.

Класс Node описывает элемент иерархического списка. Он содержит в себе поле `next`, в котором хранится указатель на следующий элемент списка, и поле `value`, в котором хранится указатель на другой список или значение типа `base`.

Совместное хранение данных в поле `value` реализовано с помощью универсального типа `variant`. Использование `variant` делает работу с классом более удобной, так как нет необходимости отдельно хранить и изменять флаг, который показывает, что хранится в поле.

Вместо обычных указателей используются умные указатели, что гарантирует корректную работу с памятью.

Функция принимает строку, в которой записан иерархический список. С помощью функции `isCorrect()` проверяется, является ли строка корректным скобочным представлением списка. Проверка проводится с помощью добавления символов открывающейся скобки в стек и удаления их из стека при встрече символа закрывающейся скобки. При корректной записи списка стек будет пуст после прохода всей строки, и не будет предпринято попыток удаления элемента из пустого стека.

Если полученная строка не является корректным представлением списка, программа выводит сообщение об этом и завершает работу.

Если же строка корректна, её элементы записываются в вектор `vec`. Для данного вектора создаётся итератор `iter`. Также создаётся умный указатель на голову списка `head`, в который записывается `nullptr`. Затем с помощью функции

`makeList()` создаётся иерархический список. Функция `makeList` принимает ссылку на итератор `iter`. Создаются указатель на голову списка `head` и указатель на текущий элемент `cur`. В оба указателя записывается `nullptr`. Затем функция обходит вектор `vec`, начиная со второго элемента, пока не встретит символ закрывающейся скобки. Обход начинается со второго элемента, так как при вызове функции `makeList` итератор указывает на символ открывающейся скобки. Если текущий символ — символ открывающейся скобки, следующий элемент — узел. Создаётся элемент списка, в поле `value` которого записывается результат работы функции `makeList` для данного положения итератора. Если текущий символ — не скобка, следующий элемент атом. Создаётся элемент списка, в поле `value` которого записывается текущий элемент вектора `vec`. В любом случае проверяется, равен ли `head nullptr`. Если указатель `head` равен `nullptr`, указатель на следующее значение записывается в `head`, указатель `cur` равен `head`. В противном случае, указатель на следующее значение записывается в поле `next` элемента, на который указывает `cur`. Затем указатель `cur` изменяется на указатель на созданный элемент.

После создания иерархического списка формируется линейный список элементов класса `Node List`. С помощью функции `countList` подсчитывается число атомов в иерархическом списке и создаётся линейный список атомов в порядке их подсчёта. Функция `countList` принимает указатель на голову иерархического списка `head`, ссылку на счётчик атомов `k`, ссылку на указатель на голову создаваемого линейного списка `list` и ссылку на указатель на его текущий элемент `listCur`. Создаётся указатель на текущее значение `cur`, в который записывается значение `head`. Затем осуществляется обход списка, пока значение `cur` не будет равно `nullptr`. Если текущий элемент — узел, для него вызывается функция `countList`. Если текущий элемент — атом, атом добавляется в линейный список `list`, значение счётчика увеличивается на один.

Затем программа выводит полученное число атомов, которое равно числу элементов в списке, и сформированный линейный список значений атомов.

Формат входных и выходных данных.

На вход программе подаётся строка, которая содержит скобочную запись иерархического списка.

Программа выводит число атомов исходного списка, а затем с новой строки линейный список значений атомов в формате : [<атом1>, <атом2>, ... , <атомN>].

Описание основных структур данных и функций.

- Класс Node – элемент иерархического списка.
- Функция makeList() – создаёт иерархический список.
- Функция isCorrect() – проверяет корректность строки со скобочной записью списка.
- Функция countList – подсчитывает число атомов и создаёт линейный список атомов.

Тестирование.

Для проведения тестирования был написан bash-скрипт tests_script. Скрипт запускает программу с определёнными входными данными и сравнивает полученные результаты с готовыми ответами. Для каждого теста выводится сообщение TestX <входные данные> - passed или TestX <входные данные> - failed. Для каждого теста выводится ожидаемый результат и полученный. Полученные в ходе работы файлы с выходными данными удаляются.

Результаты тестирования см. в приложении Б.

Выводы.

Было изучено понятие иерархического списка. Был реализован иерархический список на языке программирования C++.

Была написана программа, осуществляющая создание иерархического списка, подсчитывающая число его атомов и формирующая линейный список атомов, соответствующий порядку подсчёта. Проведено тестирование работы программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab2.cpp

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

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

template <typename base>
std::shared_ptr<Node<base>> makeList(typename
std::vector<base>::const_iterator& i){
    using NodePtr = std::shared_ptr<Node<base>>;
    NodePtr head = nullptr;
    NodePtr cur = nullptr;
    for (++i; *i != ')' ; i++){
        if ( *i == '('){
            if (head == nullptr){
                head = std::make_shared<Node<base>>();
                head->value = makeList<base>(i);
                cur = head;
            }else{
                cur->next = std::make_shared<Node<base>>();
                cur = cur->next;
                cur->value = makeList<base>(i);
            }
        }
    }
}
```

```

    }else {
        if (head == nullptr) {
            head = std::make_shared<Node<base>>();
            head->value = *i;
            cur = head;
        } else {
            cur->next = std::make_shared<Node<base>>();
            cur = cur->next;
            cur->value = *i;
        }
    }
}
return head;
}

```

```

bool isCorrect(const std::string& str){
    std::stack<char> Stack;
    if (str[0] != '('){
        return false;
    }
    for (char i : str){
        if (i == '('){
            Stack.push(i);
        }
        if (i == ')'){
            if (Stack.empty()){
                return false;
            }
            Stack.pop();
        }
    }
    return Stack.empty();
}

```

```

template <typename base>

```

```

void countList(std::shared_ptr<Node<base>> head, int& k,
std::shared_ptr<Node<base>>& listHead,
std::shared_ptr<Node<base>>& listCur){
    using NodePtr = std::shared_ptr<Node<base>>;
    NodePtr cur = head;
    while(cur != nullptr) {
        if (std::holds_alternative<NodePtr>(cur->value)) {
            countList(std::get<NodePtr>(cur->value), k, listHead,
listCur);
        } else {
            k++;
            if (listHead == nullptr){
                listHead = std::make_shared<Node<base>>();
                listHead->value = std::get<char>(cur->value);
                listCur = listHead;
            }else{
                listCur->next = std::make_shared<Node<base>>();
                listCur = listCur->next;
                listCur->value = std::get<char>(cur->value);
            }
        }
        cur = cur->next;
    }
}

```

```

int main()
{
    std::string listString;
    std::cin >> listString;
    if (!isCorrect(listString)){
        std::cout << "List is not correct\n";
        return 0;
    }
    std::vector<char> vec(listString.begin(), listString.end());
    std::shared_ptr<Node<char>> head = nullptr;
    auto iter = vec.cbegin();
}

```

```

head = makeList<char>(iter);
int counter = 0;
std::shared_ptr<Node<char>> list = nullptr;
std::shared_ptr<Node<char>> listCur = nullptr;
countList(head, counter, list, listCur);
std::cout << counter << "\n";
listCur = list;
if (counter > 0){
    std::cout << "[";
    while (listCur->next != nullptr){
        std::cout << std::get<char>(listCur->value) << ", ";
        listCur = listCur->next;
    }
    listHead = std::make_shared<Node<base>>();
    listHead->value = std::get<char>(cur->value);
    listCur = listHead;
} else {
    listCur->next = std::make_shared<Node<base>>();
    listCur = listCur->next;
    listCur->value = std::get<char>(cur->value);
}
}
cur = cur->next;
}

int main()
{
    std::string listString;
    std::cin >> listString;
    if (!isCorrect(listString)){
        std::cout << "List is not correct\n";
        return 0;
    }
    std::vector<char> vec(listString.begin(), listString.end());
    std::shared_ptr<Node<char>> head = nullptr;

```



```

auto iter = vec.cbegin();
head = makeList<char>(iter);
int counter = 0;
std::shared_ptr<Node<char>> list = nullptr;
std::shared_ptr<Node<char>> listCur = nullptr;
countList(head, counter, list, listCur);
std::cout << counter << "\n";
listCur = list;
if (counter > 0){
    std::cout << "[";
    while (listCur->next != nullptr){
        std::cout << std::get<char>(listCur->value) << ", ";
        listCur = listCur->next;
    }
    std::cout << std::get<char>(listCur->value) << "]" <<
std::endl;
}else{
    std::cout << "[]" << std::endl;
}
return 0;
}

```

Название файла: tests_script

```
#!/bin/bash
```

```

printf "\nRunning tests...\n\n"
for n in {1..8}
do
    ./lab2 < "./Tests/tests/test$n.txt" > "./Tests/out/out$n.txt"
    printf "Test$n: "
    cat "./Tests/tests/test$n.txt" | tr -d '\n'
    if cmp "./Tests/out/out$n.txt"
"./Tests/true_results/true_out$n.txt" > /dev/null; then
        printf " - Passed\n"
    else

```

```
        printf " - Failed\n"
    fi
    printf "Desired result:\n"
    cat "./Tests/true_results/true_out$n.txt"
    printf "Actual result:\n"
    cat "./Tests/out/out$n.txt"
    printf "\n"
done
rm ./Tests/out/out*
```

Название файла: Makefile

```
lab2: Source/lab2.cpp
    g++ Source/lab2.cpp -o lab2

run_tests: lab2
    ./tests_script
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

№	Входные данные	Выходные данные	Комментарии
1	(ab(c(df)g)e(a))	8 [a, b, c, d, f, g, e, a]	Корректный список, содержащий узлы и атомы
2	()	0 []	Пустой список
3	((00000000((((0))))))	0 []	Список, содержащий только узлы
4	(a(b)()c()d()e()(f((g((e))v))x))	10 [a, b, c, d, e, f, g, e, v, x]	Корректный список, содержащий узлы и атомы
5	(abcdefge)	8 [a, b, c, d, e, f, g, e]	Список, содержащий только атомы
6	(a(b(c(d(e(f(g(x)y)z)k)i)j)lm))	15 [a, b, c, d, e, f, g, x, y, z, k, i, j, l, m]	Корректный список, содержащий узлы и атомы
7	(a(bc)e())	List is not correct	Некорректный список
8	()(ace(acde(()ad))	List is not correct	Некорректный список

