**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

**отчет**

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Иерархические списки**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9304 |  | Селезнёва А.В. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2020

**Цель работы.**

Ознакомиться с понятием иерархического списка. Реализовать иерархический список для решения поставленной задачи на языке программирования С++.

**Задание.**

Вариант – 24.

Пусть алгебраическое выражение представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<*операция*> <*аргументы*>)).

Доступные операции: +, -, \*, *power(,)*.

На входе дополнительно задаётся список значений переменных

где – переменная, а – её значение (константа).

Требуется выполнить вычисление введенного выражения.

**Выполнение работы.**

На вход программа получает две строки, каждая из которых содержит скобочную запись иерархического списка. Первая строка является алгебраическим выражением, а вторая списком значений переменных. Далее на основе этих строк создаются два иерархических списка. Пример хранения введенных данных «(\* 6(- c 4))» и «((c 3))» в иерархических списках представлен на рисунке 1 и на рисунке 2 соответственно:

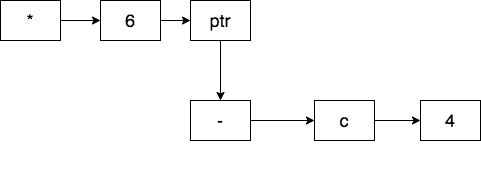
****

Рисунок 1 – Иллюстрация хранения алгебраического выражения в иерархическом списке

****

Рисунок 2 – Иллюстрация списка значений элементов в иерархическом списке

Далее вызывается метод *call\_cal()* для вычисления алгебраического выражения. Результат работы этого метода выводится на экран.

Класс *Node*:

Класс содержит поле *Next* – умный указатель на следующий элемент списка. Поле *elem* – *std::varian*, который может содержать строку или умный указатель на иерархический список.

Класс *List\_Hier*:

Класс содержит поле *Head* – умный указатель на первый элемент иерархического списка.

Конструктор класса принимает строку, проверяет ее на корректность и в случае ее корректности создает иерархический список, иначе выбрасывает исключение.

Метод *call\_cal()* на вход получает список элементов, отправляет их в метод *calculating()* вместе с указателем на первый элемент списка и возвращает результат метода *calculating()*.

Метод *calculating()* рекурсивно вычисляет алгебраическое выражение, возвращает число – результат вычислений. Данный метод выбрасывает исключение в случае ошибки.

Метод *Search\_for\_an\_element()* рекурсивно обходит список и находит в нем строку, равную строке, полученную в вызове метода, и возвращает число, которое соответствует числу данного элемента. Также метод выбрасывает исключение в зависимости от ошибки.

Метод *App\_hier()* добавляет в конец иерархического списка полученный *Node*.

Разработанный программный код находится в приложении А.

**Тестирование.**

Тестирование осуществляется с помощью bash-скрипта ./script. Скрипт запускает программу и в качестве входных аргументов подает строки, прописанные в текстовых файлах, расположенных в папке ./Tests.

Результаты тестирования представлены в приложении Б.

**Выводы.**

В ходе выполнения лабораторной работы был реализован иерархический список на языке программирования С++.

Разработана программа, создающая иерархический список и вычисляющая алгебраическое выражение с его помощью. Использование иерархического списка при решении поставленной задачи не оправдано с точки зрения экономии памяти, так как используется рекурсивная обработка самого иерархического списка.

**Приложение А**

**Исходный код программы**

Название файла: Lb2.cpp

#include "List\_Hier.h"

#include <iostream>

int main() {

std::string s\_elem;

std::string s\_cal;

std::getline(std::cin, s\_cal);

std::getline(std::cin, s\_elem);

try {

List\_Hier list\_cal(s\_cal);

List\_Hier list\_elem(s\_elem);

std::cout << list\_cal.call\_cal(list\_elem);

}

catch (const char\* ex) {

std::cout << ex;

}

return 0;

}

Название файла: List\_Hier.h

#ifndef LIST\_HEIR

#define LIST\_HEIR

#include <variant>

#include <string>

#include <cmath>

#include <cctype>

#include <cstdlib>

#include <memory>

class List\_Hier;

class Node {

friend class List\_Hier;

std::variant<std::string, std::shared\_ptr<List\_Hier>> elem;

std::shared\_ptr<Node> Next = nullptr;

};

class List\_Hier

{

public:

std::shared\_ptr <Node> Head = nullptr;

List\_Hier(const std::string& str);

int call\_cal( List\_Hier& List\_elements);

void App\_hier(const Node& new\_Node);

int calculating(std::shared\_ptr <Node> ptrNode, List\_Hier& List\_elements);

int Search\_for\_an\_element(std::shared\_ptr <Node> ptr, const std::string str);

};

#endif

Название файла: List\_Hier.cpp

#include "List\_Hier.h"

size\_t search\_the\_bracket(const std::string str) {

size\_t i = 0;

int bracket = 0;

do {

if (str.at(i) == '(') {

++bracket;

}

else if (str.at(i) == ')') {

--bracket;

}

++i;

} while (bracket != 0 && i < str.size());

if (bracket != 0) {

return 0;

}

return i;

}

List\_Hier::List\_Hier(const std::string& str)

{

bool is\_first\_bracket = true;

size\_t i = 0;

while (i < str.size()) {

if (str.at(i) == '(') {

if (search\_the\_bracket(str) == 0 && search\_the\_bracket(str) != str.size()) {

throw ("input not true\n");

}

if (is\_first\_bracket) {

is\_first\_bracket = false;

++i;

}

else {

std::shared\_ptr <List\_Hier> new\_elem(new List\_Hier(str.substr(i, str.size())));

Node new\_Node;

new\_Node.elem = new\_elem;

this->App\_hier(new\_Node);

i = str.size();

}

}

else if (isdigit(str.at(i)) || isalpha(str.at(i)) || str.at(i) == '+' || str.at(i) == '-' || str.at(i) == '\*') {

if (is\_first\_bracket) {

throw ("input not true\n");

}

else {

if (str.find("power(", i) == i) {

if (str.find(',', i) == -1) {

throw ("input not true\n");

}

else {

const std::string new\_elem = str.substr(i, 5);

Node new\_Node;

new\_Node.elem = (std::string)new\_elem;

this->App\_hier(new\_Node);

i = i + 6;

size\_t comma = str.find(',', i);

std::shared\_ptr <List\_Hier> new\_elem\_1(new List\_Hier(str.substr(i, comma - i)));

Node new\_Node\_1;

new\_Node\_1.elem = new\_elem\_1;

this->App\_hier(new\_Node\_1);

size\_t end\_two\_arg = search\_the\_bracket(str.substr(comma + 1, str.size()));

std::shared\_ptr <List\_Hier> new\_elem\_2(new List\_Hier(str.substr(comma + 1, end\_two\_arg - comma - 2)));

Node new\_Node\_2;

new\_Node\_2.elem = new\_elem\_2;

this->App\_hier(new\_Node\_2);

i = end\_two\_arg+1;

}

}

else {

size\_t i\_1 = i;

i = str.find\_first\_not\_of("1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM-+\*", i + 1);

const std::string new\_elem = str.substr(i\_1, i - i\_1);

Node new\_Node;

new\_Node.elem = (std::string)new\_elem;

this->App\_hier(new\_Node);

}

}

}

else if (str.at(i) == ' ' || str.at(i) == ')' || str.at(i) == ',') {

++i;

}

else {

throw ("input not true\n");

}

}

}

void List\_Hier::App\_hier(const Node& new\_Node) {

std::shared\_ptr <Node> node = std::make\_shared<Node>(new\_Node);

std::shared\_ptr <Node> ptr = this->Head;

if (this->Head != nullptr) {

while (ptr->Next != nullptr) {

ptr = ptr->Next;

}

ptr->Next = node;

}

else {

this->Head = node;

}

}

bool from\_string\_to\_int(const std::string str) {

size\_t i = 0;

if (str.at(0) == '-' || str.at(0) == '+') {

++i;

}

while (i < str.size()) {

if (isdigit(str.at(i))) {

++i;

}

else {

return false;

}

}

return true;

}

int List\_Hier::call\_cal( List\_Hier& List\_elements) {

return calculating(this->Head, List\_elements);

}

int List\_Hier::calculating(std::shared\_ptr <Node> ptr\_Node, List\_Hier& List\_elements) {

if (ptr\_Node == nullptr) {

throw ("error calculating\n");

}

else if (std::holds\_alternative<std::string>(ptr\_Node->elem)) {

std::string str = std::get<std::string>(ptr\_Node->elem);

if (str == "+") {

if (ptr\_Node->Next == nullptr && ptr\_Node->Next->Next == nullptr) {

throw ("error calculating\n");

}

int first = calculating(ptr\_Node->Next, List\_elements);

int second = calculating(ptr\_Node->Next->Next, List\_elements);

return first + second;

}

else if (str == "-") {

if (ptr\_Node->Next == nullptr && ptr\_Node->Next->Next == nullptr) {

throw ("error calculating\n");

}

int first = calculating(ptr\_Node->Next, List\_elements);

int second = calculating(ptr\_Node->Next->Next, List\_elements);

return first - second;

}

else if (str == "\*") {

if (ptr\_Node->Next == nullptr && ptr\_Node->Next->Next == nullptr) {

throw ("error calculating\n");

}

int first = calculating(ptr\_Node->Next, List\_elements);

int second = calculating(ptr\_Node->Next->Next, List\_elements);

return first \* second;

}

else if (str == "power") {

if (ptr\_Node->Next == nullptr && ptr\_Node->Next->Next == nullptr) {

throw ("error calculating\n");

}

int first = calculating(ptr\_Node->Next, List\_elements);

int second = calculating(ptr\_Node->Next->Next, List\_elements);

return (int)pow(first, second);

}

else {

if (from\_string\_to\_int(str)) {

if (str.at(0) == '-') {

std::string str\_1 = str.substr(1, str.size() - 1);

return -atoi(str\_1.c\_str());

}

else if (str.at(0) == '+') {

std::string str\_1 = str.substr(1, str.size() - 1);

return atoi(str\_1.c\_str());

}

else {

return atoi(str.c\_str());

}

}

else {

return Search\_for\_an\_element(List\_elements.Head, str);

}

}

}

else if (std::holds\_alternative<std::shared\_ptr<List\_Hier>>(ptr\_Node->elem)) {

return calculating(std::get< std::shared\_ptr<List\_Hier>>(ptr\_Node->elem)->Head, List\_elements);

}

throw ("error calculating\n");

}

int List\_Hier::Search\_for\_an\_element(std::shared\_ptr <Node> ptr, const std::string str) {

if (ptr == nullptr) {

throw ("list of elements is empty\n");

}

while (ptr != nullptr) {

if (std::holds\_alternative<std::string>(ptr->elem)) {

std::string str\_node = std::get<std::string>(ptr->elem);

if (str\_node == str) {

std::string str\_elem = std::get<std::string>(ptr->Next->elem);

if (from\_string\_to\_int(str\_elem)) {

if (str\_elem.at(0) == '-') {

std::string str\_1 = str\_elem.substr(1, str.size() - 1);

return -atoi(str\_1.c\_str());

}

else {

return atoi(str\_elem.c\_str());

}

}

else {

throw ("element is not an integer\n");

}

}

}

else if (std::holds\_alternative<std::shared\_ptr<List\_Hier>>(ptr->elem)) {

return Search\_for\_an\_element(std::get< std::shared\_ptr<List\_Hier>>(ptr->elem)->Head, str);

}

ptr = ptr->Next;

}

throw ("element is not found\n");

}

**Приложение Б**

**Тестирование**

Таблица Б – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Результат проверки |
| 1. | (- a(+ c(v)))  ((a 3)(c 10)(v 11)) | -18 | correct |
| 2. | (\* 6(- c 4))  ((c 3)) | -6 | correct |
| 3. | (power((+ 7(m)),(\* l(4))))  ((m 3)(l 2)) | 100 000 000 | correct |
| 4. | (+ 6(a))  ((c 4)) | element is not found | correct |
| 5. | (- \*(d))  ((d 4)) | error calculating | correct |
| 6. | (- a(+ 3(-4)))  ((a 12)) | 13 | correct |
| 7. | (- 4(power((\* b(2)),(+7(c)))))  ((c 2)(b 4)) | -134 217 724 | correct |
| 8. | (\* a(+ c(b)))  ((a 5)(b 2)(c 3)) | 25 | correct |