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: Рекурсивная обработка иерархических списков Вариант 10

Студент гр. 8304 Сани З. Б.

Преподаватель Фиалковский М. С.

Санкт-Петербург 2019

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, с использованием базовых функций их рекурсивной обработки.

Постановка задачи.

1. проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2. разработать программу, использующую рекурсию;
3. сопоставить рекурсивное решение с итеративным решением задачи;
4. сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

Вариант 10: подсчитать число различных атомов в иерархическом списке; сформировать из них линейный список;

Описание алгоритма.

Для начала программа должна рекурсивно считать данные, проверить их корректность и занести их в список. Для этого считываем очередной символ строки. Если это атом, добавляем его к иерархическому списку, если это список– рекурсивно заносим его в список.

Далее, Если текущий список - атом, тогда добавляем данные в множество, иначе вызываем функцию рекурсивно для вложенного и следующего списка, затем мы проверяем размер множество, чтобы получить количество различных атомов, и мы проходим через набор, чтобы сформировать линейный список

Анализ алгоритма.

Алгоритм работает за линейное время от размера списка. Недостаток рекурсивного алгоритма – ограниченный стек вызовов функций, что в свою

очередь накладывает ограничение на количество вложенных списков, а также затраты производительности на вызов функций.

Описание функций и СД.

Класс-реализация иерархического списка ListPointer содержит умный указатель на вложенный список (Nested), умный указатель на следующий элемент списка (NextList), флаг, для определения атома, значение атома. Умные указатели были использованы во избежание утечек памяти.

Статический метод класса для проверки входных данных на корректность:

```
static bool isCorrectStr(const std::string& str);
```

Принимает на вход ссылку на строку, проверяет размер и структуру строки, возвращает true, если строка корректна, и false в ином случае.

Статический метод класса для создания иерархического списка:

```
static bool buildList(LisPointer& list, const std::string& str);
```

Принимает на вход ссылку на строку и ссылку на умный указатель на список, проверяет корректность строки и вызывает функции рекурсивного создания списка.

Приватный метод класса для считывания вложенных списков:

```
static void readData(ListPointer& list, const char prev,
std::string::const_iterator& it, const std::string::const_iterator&
end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент строки и итераторы на строку. Если предыдущий элемент не равен “(“, создается атом, в ином случае вызывает считывание следующего списка.

Приватный метод класса для считывания следующих списков:

```
static void readSeq(ListPointer& list, std::string::const_iterator& it, const std::string::const_iterator&
end);
```

Принимает на вход ссылку на умный указатель на список, предыдущий элемент

строки и итераторы на строку. Если строка пустая – происходит return. Если элемент равен “)”, создается пустой список. В ином случае рекурсивно считываются вложенные и следующие списки, затем объединяются в текущем спис

ПРИЛОЖЕНИЕ

1. **Тестирование.**

Работа программы для строки (a(a())b)

```
> Choose your input
> 0 - from console
> 1 - from file default file -(default test file is located along the path : test2.txt)
> Any other key to Exit!
0
> Enter List: (a(a())b)
В список добавляется содержимое следующих скобок:(a(a())b)
Проверка на корректность:(a(a())b)
Строка корректна.
Число различных элементов в списке: 2
Линейный список: (ab)

Program ended with exit code: 0
```

Структура иерархического списка

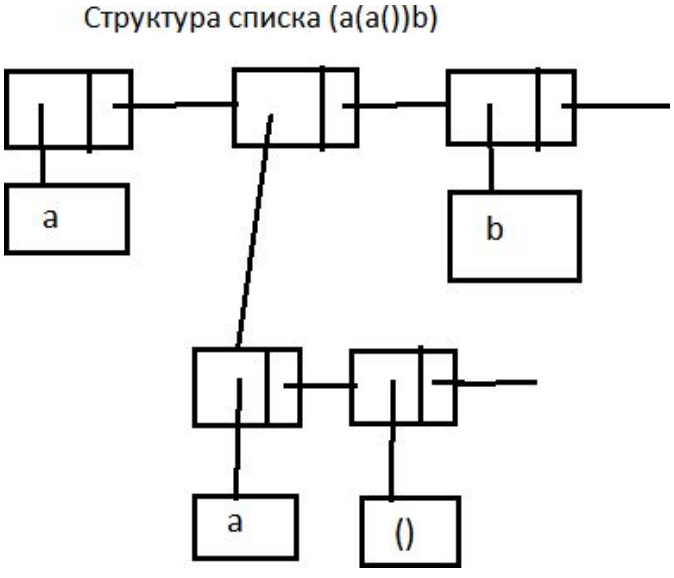

Таблица результатов ввода/вывода тестирования программы

введенный список	**Корректность**	**Количество**	**Линейный список**
()	корректно	0	()
(a)	корректно	1	(a)
(abc)	корректно	3	(abc)
(a(bc)de)	корректно	5	(abcde)
(a(bc(d))o)	корректно	5	(abcdo)
(a(a())b)	корректно	2	(ab)
(ab()(a()))	корректно	2	(ab)
((a)(cd(jaa)(hj)))	корректно	5	(achdj)
(a()()b()c()d(efga(cyda)))	корректно	8	(abcydefg)
((a)(b(cd)))	корректно	4	(abcd)
)abd(	некорректно	-	-
(t(yi(syd))	некорректно	-	-
(t(yi(syd)))	корректно	5	(tyisd)
(a(bc)((de(f))g)h)	корректно	8	(abcdefgh)
(abcd))	некорректно	-	-

```
> Choose your input
> 0 - from console
> 1 - from file default file -(default test file is located along the path : test2.txt)
> Any other key to Exit!
1
> FilePath: /Users/sanizayyad/Documents/Sani_Zayyad/lab2/Test/test2.txt
Reading from file:

test #1 "()"
В список добавляется содержимое следующих скобок:()
Проверка на корректность:()
Строка корректна.
Число различных элементов в списке: 0
Линейный список: ()

test #2 "(a)"
В список добавляется содержимое следующих скобок:(a)
Проверка на корректность:(a)
Строка корректна.
Число различных элементов в списке: 1
Линейный список: (a)

test #3 "(abc)"
В список добавляется содержимое следующих скобок:(abc)
Проверка на корректность:(abc)
Строка корректна.
Число различных элементов в списке: 3
Линейный список: (abc)

test #4 "(a(bc)de)"
В список добавляется содержимое следующих скобок:(a(bc)de)
Проверка на корректность:(a(bc)de)
Строка корректна.
Число различных элементов в списке: 5
Линейный список: (abcde)

test #5 "(a(bc(d))o)"
В список добавляется содержимое следующих скобок:(a(bc(d))o)
Проверка на корректность:(a(bc(d))o)
Строка корректна.
Число различных элементов в списке: 5
Линейный список: (abcdo)

test #6 "(a(a())b)"
В список добавляется содержимое следующих скобок:(a(a())b)
Проверка на корректность:(a(a())b)
Строка корректна.
Число различных элементов в списке: 2
Линейный список: (ab)
```

```
test #7 "(ab()(a()))"
В список добавляется содержимое следующих скобок:(ab()(a()))
Проверка на корректность:(ab()(a()))
Строка корректна.
Число различных элементов в списке: 2
Линейный список: (ab)

test #8 "((a)(cd(jaa)(hj)))"
В список добавляется содержимое следующих скобок:((a)(cd(jaa)(hj)))
Проверка на корректность:((a)(cd(jaa)(hj)))
Строка корректна.
Число различных элементов в списке: 5
Линейный список: (achdj)

test #9 "(a()()b()c()d(efga(cyda)))"
В список добавляется содержимое следующих скобок:(a()()b()c()d(efga(cyda)))
Проверка на корректность:(a()()b()c()d(efga(cyda)))
Строка корректна.
Число различных элементов в списке: 8
Линейный список: (abcydefg)

test #10 "((a)(b(cd)))"
В список добавляется содержимое следующих скобок:((a)(b(cd)))
Проверка на корректность:((a)(b(cd)))
Строка корректна.
Число различных элементов в списке: 4
Линейный список: (abcd)

test #11 ")abd("
В список добавляется содержимое следующих скобок:)abd(
Проверка на корректность:)abd(
Строка Некорректна.
test #12 "(t(yi(syd))"
В список добавляется содержимое следующих скобок:(t(yi(syd))
Проверка на корректность:(t(yi(syd))
Строка Некорректна.
test #13 "(t(yi(syd)))"
В список добавляется содержимое следующих скобок:(t(yi(syd)))
Проверка на корректность:(t(yi(syd)))
Строка корректна.
Число различных элементов в списке: 5
Линейный список: (tyisd)

test #14 "(a(bc)((de(f))g)h)"
В список добавляется содержимое следующих скобок:(a(bc)((de(f))g)h)
Проверка на корректность:(a(bc)((de(f))g)h)
Строка корректна.
Число различных элементов в списке: 8
Линейный список: (abcdefgh)

test #15 "(abcd))"
В список добавляется содержимое следующих скобок:(abcd))
Проверка на корректность:(abcd))
Строка Некорректна.
Program ended with exit code: 0
```

Выводы.

В ходе работы я вспомнил работу с классами и контейнерами в C++. Научился решать задачи обработки иерархических списков, с использованием базовых функций их рекурсивной обработки и сравнивать два иерархических списка на структурную идентичность.

2. Исходный код программы.

hierachy.h

```
#ifndef HIERARCHICALLIST_H
#define HIERARCHICALLIST_H

#include <memory>
#include <iostream>


class HierarchicalList
{
public:
    //Умный указатель на список для избежания утечек памяти
    typedef std::shared_ptr<HierarchicalList> ListPointer;

    explicit HierarchicalList();
    ~HierarchicalList() = default;

     //Функции для доступа к данным, Nested - указатель на вложенные списки, Next - на следующие
    ListPointer getNested() const;
    ListPointer getNext() const;
    bool isNull() const;
    bool isAtom() const;
    char getAtom() const;

    //static-методы можно вызывать как обычные функции, не создавая объект класса
    static bool buildList(ListPointer& list, const std::string& str);

     //Перегрузка оператора cout для печати списка, и функция рекурсивной печати следующих списков
    friend std::ostream& operator<< (std::ostream& out, const ListPointer& list);
```

```
    void print_seq(std::ostream& out) const;

private:
    //Функция проверки входной строки на корректность
    static bool isCorrectStr(const std::string& str);

    //функции рекурсивного чтения списка
    static void readData(ListPointer& list, const char prev, std::string::const_iterator& it,
                         const std::string::const_iterator& end);
    static void readSeq(ListPointer& list, std::string::const_iterator& it,
                        const std::string::const_iterator& end);

    //Функция конструирования непустого списка
    static ListPointer cons(ListPointer& nestedList, ListPointer& nextList);

    //превращает объект класса в атом(список с данными)
    void createAtom(const char ch);


private:
    //Флаг для определения, является ли список - атомом
    bool flag;
    //указатели на вложенный и следующий список
    ListPointer nestedList;
    ListPointer nextList;
    //данные
    char data;
};

#endif // HIERARCHICALLIST_H
```

hierachy.cpp

```
#include "hierachy.h"

HierarchicalList::HierarchicalList()
{
    /*
     * По умолчанию объект класса является пустым списком
    */

    flag = false;
    data = 0;

    nextList.reset();
```

```
    nestedList.reset();
}


bool HierarchicalList::isNull() const
{
    /*
     * Возвращает true, если элемент является нулевым списком,
     * false - в ином случае
    */

    return (!flag && nestedList == nullptr && nextList == nullptr);
}


HierarchicalList::ListPointer HierarchicalList::getNext() const
{
    /*
     * Если элемент не атом - возращает указатель на вложенный список,
     * в ином случае - nullptr
    */

    if (!flag) {
        return nextList;
    }
    else {
        std::cerr << "Error: Next(atom)\n";
        return nullptr;
    }
}


HierarchicalList::ListPointer HierarchicalList::getNested() const
{
    /*
     * Если элемент не атом - возращает указатель на следующий список,
     * в ином случае - nullptr
    */

    if (!flag) {
        return nestedList;
    }
    else {
        std::cerr << "Error: Nested(atom)\n";
        return nullptr;
    }
```

```
}


bool HierarchicalList::isAtom() const
{
   /*
    * Если элемент атом - возращает true,
    * в ином случае - false
   */

   return flag;
}


HierarchicalList::ListPointer HierarchicalList::cons(ListPointer& nestedList, ListPointer& nextList)
{
   /*
    * Функция создания списка
   */

   if (nextList != nullptr && nextList->isAtom()) {
      std::cerr << "Error: Next(atom)\n";
      return nullptr;
   }
   else {
      ListPointer tmp(new HierarchicalList);
      tmp->nestedList = nestedList;
      tmp->nextList = nextList;
      return tmp;
   }
}


char HierarchicalList::getAtom() const
{
   /*
    * Функция возвращает значение атома
   */
   if (flag) {
      return data;
   }
   else {
      std::cerr << "Error: getAtom(!atom)\n";
      return 0;
   }
```

```
}


void HierarchicalList::print_seq(std::ostream& out) const
{
    /*
     * Функция печати Tail
    */

    if (!isNull()) {
        out << this->getNested();

        if (this->getNext() != nullptr)
            this->getNext()->print_seq(out);
    }
}


std::ostream& operator<<(std::ostream& out, const HierarchicalList::ListPointer& list)
{
    /*
     * Перегрузка оператора вывода
    */

    if (list == nullptr || list->isNull()) {
        out << "()";
    }
    else if (list->isAtom()) {
        out << list->getAtom();
    }
    else {
        out << "(";

        out << list->getNested();
        if (list->getNext() != nullptr)
            list->getNext()->print_seq(out);

        out << ")";
    }

    return out;
}


bool HierarchicalList::isCorrectStr(const std::string& str)
{
```

```
    /*
     * Функция проверки корректности входных данных,
     * принимает на вход ссылку на строку, проверяет размер и структуру строки,
     * возвращает true, если строка корректна, и false в ином случае
    */

    std::cout << "Проверка на корректность:" << str << "\n";
    int countBracket = 0;

        if (str.size() < 2){
            std::cout << "Строка Некорректна.\n";
            return false;
        }

    if (str[0] != '(' || str[str.size() - 1] != ')'){
        std::cout << "Строка Некорректна.\n";
        return false;
    }

    size_t i;
    for (i = 0; i < str.size(); ++i) {
        char elem = str[i];
        if (elem == '('
            ++countBracket;
        else if (elem == ')')
            --countBracket;
        else if (!isalpha(elem)){
            std::cout << "Строка Некорректна.\n";
            return false;
        }

        if (countBracket <= 0 && i != str.size()-1){
            std::cout << "Строка Некорректна.\n";
            return false;
        }
    }

    if (countBracket != 0 || i != str.size()) {
        std::cout << "Строка Некорректна.\n";
        return false;
    }

    std::cout << "Строка корректна.\n";
    return true;
}
```

```
void HierarchicalList::createAtom(const char ch)
{
   /*
    * Создается объект класса - атом
   */
   this->data = ch;
   this->flag = true;
}


void HierarchicalList::readData(ListPointer& list, const char prev, std::string::const_iterator &it,
                        const std::string::const_iterator& end)
{
   /*
       * Функция считывания данных. Считывает либо атом, лио рекурсивно считывает
список
   */

   if (prev != '(') {
      list->createAtom(prev);
   }
   else {
      readSeq(list, it, end);
   }
}


void HierarchicalList::readSeq(ListPointer& list, std::string::const_iterator&it,
                        const std::string::const_iterator& end)
{
   /*
    * Функция считывания списка. Рекусривно считывает данные и спискок и
    * добавляет их в исходный.
   */

   ListPointer nestedList(new HierarchicalList);
   ListPointer nextList(new HierarchicalList);

   if (it == end)
      return;

   if (*it == ')') {
      ++it;
   }
   else {
```

```
        char prev = *it;
        ++it;
        readData(nestedList, prev, it, end);
        readSeq(nextList, it, end);
        list = cons(nestedList, nextList);
    }
}


bool HierarchicalList::buildList(ListPointer& list, const std::string& str)
{
    /*
     * Функция создания иерархического списка. Принимает на вход ссылку
     * на строку, проверяет корректность строки и вызывает приватный метод
     * readData().
    */

    std::cout << "В список добавляется содержимое следующих скобок:" << str << "\n";

    if (!isCorrectStr(str))
        return false;

    auto it_begin = str.cbegin();
    auto it_end = str.cend();
    char prev = *it_begin;
    ++it_begin;
    readData(list, prev, it_begin, it_end);

    return true;
}
```

main.cpp

```
#include "hierachy.h"
#include <iostream>
#include <unordered_set>
#include <list>
#include <fstream>

void toSet(std::unordered_set<char>& set, const HierarchicalList::ListPointer list);
void ReadFromFile(std::string filename);
void execute(std::string listStr);
```

```
void toSet(std::unordered_set<char>& set, const HierarchicalList::ListPointer list) {
   if (list != nullptr && !list->isNull()) {
      if (list->isAtom()) {
         set.insert(list->getAtom());
      }
      else {
         toSet(set, list->getNested());
         toSet(set, list->getNext());
      }
   }
}


void ReadFromFile(std::string filename)
{
   std::ifstream file(filename);

   if (file.is_open())
   {
      std::cout << "Reading from file:"<< "\n\n";
      int count = 0;
      while (!file.eof())
      {
         count++;
         std::string listStr;
         getline(file, listStr);
         std::cout << "test #" << count << " \"" + listStr + "\""<< "\n";
         execute(listStr);
      }
   }
   else
   {
      std::cout << "File not opened"<<"\n";
   }
}

void execute(std::string listStr){
   HierarchicalList::ListPointer list(new HierarchicalList);
   std::unordered_set<char> set;

   if (HierarchicalList::buildList(list, listStr)) {
      toSet(set, list);
      std::cout << "Число различных элементов в списке: " << set.size() << "\n";

      std::list<char> result;
      for (auto elem : set)
```

```
            result.push_front(elem);

        std::string resultStr;
        for (auto elem : result)
            resultStr += elem;
        std::cout << "Линейный список: (" << resultStr << ")\n\n";
    }

}



int main()
{
    std::cout << "> Choose your input" << "\n";
    std::cout << "> 0 - from console" << "\n";
     std::cout << "> 1 - from file default file -(default test file is located along the path : test2.txt)"
<< "\n";
    std::cout << "> Any other number to Exit!" <<"\n";

    int command = 0;
    std::cin >> command;

    switch (command)
    {
        case 0:
        {
        std::string input;
        std::cout << "> Enter List: ";
        std::cin>>input;
        execute(input);
        break;
        }
    case 1:
        {
        std::cout << "> FilePath: ";
        std::string filePath;
        std::cin >> filePath;
        ReadFromFile(filePath);
        break;

        }
    default:
        std::cout << "GOODBYE!";
    }
    return 0;
```

}