МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 9303 _____ Низовцов Р. С.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием «иерархический список», изучить его особенности и реализовать программу, решающую поставленную задачу с помощью иерархического списка.

Задание.

Вариант 17

Пусть выражение (логическое) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме **((<операция> <аргументы>))**. Аргументов может быть 1, 2 и более. Например (в префиксной форме):

(OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: упрощение (преобразование). Пример упрощения:

(+ 0 (* 1 (+ a b))) преобразуется в (+ a b).

Основные теоретические положения.

Иерархические списки состоят из элементов различных уровней, при этом элементы нижних уровней подчинены элементам верхних уровней. Существует два вида иерархии списков: иерархия групп и элементов и иерархия элементов. Вид устанавливается конфигурацией.

В списке с иерархией групп и элементов содержатся два вида элементов – группы и собственно элементы. Группа обозначает узел, в который входят другие (подчиненные) группы и элементы, а элемент является конкретным объектом.

Для списков с иерархией элементов любой из элементов может быть как узлом, так и отдельным объектом. Примером может служить список подразделений. Каждое подразделение может содержать в своем составе другие подразделения, но набор свойств у всех подразделений будет одинаков.

Выполнение работы.

Для выполнения программы были реализованы функции readList, simplifyList, writeList.

В функции int main() запрашивается путь к файлу с данными для обработки, проверяется её корректность. Потом запускается цикл и работает, пока getLine не вернет false. После этого информирует об окончании работы, закрывает все файлы.

В функции void readList(string) проводится посимвольное чтение из line до знака „)” , знака конца строки. Из них создается иерархический список.

В функции void simplifyList() проводится упрощение логических выражений: сначала удаляются повторы аргументов внутри каждой операции, потом идет проверка на сочетания операций, после идет работа с аргументами TRUE и FALSE.

В функции void writeList(ofstream&) проводится вывод списка в файл, параллельно выводя результат в консоль.

Выводы.

Ознакомился с понятием «иерархический список», изучил его особенности и реализовал программу, решающую поставленную задачу с помощью иерархического списка.

Была реализована программа, включающая в себя функцию simplifyList для обработки строк, хранящихся в списке. Программа выполняет чтение, запись результата в файл, а также производит проверку и обработку считанного текста.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class base{

public:
    base *parent = nullptr;
    base *child = nullptr;
    base *prev = nullptr;
    base *next = nullptr;
    string value = "";
};

class hlist{

public:
    base *first;
    base *last;

    hlist();
    ~hlist();
    void deleteFrom(base*);
    void addNext(string);
    void addChild(string);
    void readList(string);
    void simplifyList();
    void writeList(ofstream&);
    void repeatCheck(base*);
    base* notCheck(base*, base*);
    base* andOrCheck(base*, base*);
    bool trueFalseCheck(base*);
    bool accFinder(base*, base*);
};

hlist::hlist(){
    first = new base;
    last = first;
    base *temp = new base;
    temp->child = first;
    first->parent = temp;
}

hlist::~hlist(){
    base *curr = last;
    while(curr!=first){
        if(curr->prev){
            curr = curr->prev;
```

```
            delete curr->next;
            continue;
        }
        if(curr->parent){
            curr = curr->parent;
            delete curr->child;
            continue;
        }
    }
    if(first->parent)
        delete first->parent;
    delete first;
}

void hlist::deleteFrom(base *start){
    while(start->next || start->child){
        if(start->next)
            start = start->next;
        else
            start = start->child;
        if(start->prev)
            delete start->prev;
        else
            delete start->parent;
    }
    delete start;
}

void hlist::addNext(string word){
    if(this->last->value == ""){
        this->first->value = word;
        last = first;
    }
    else{
        base * temp = new base;
        temp->value = word;
        this->last->next = temp;
        temp->prev = last;
        last = temp;
    }
}

void hlist::addChild(string word){
    if(this->last->value == ""){
        this->first->value = word;
        last = first;
    }
    else{
        base * temp = new base;
        temp->value = word;
        this->last->child = temp;
        temp->parent = last;
        last = temp;
    }
}

void hlist::readList(string line){
    bool isChild = false;
```

```
    int currChar = 1;
    string currWord = "";
    while(line[currChar] != ')' && line[currChar] != '\n'){
        if(line[currChar] == ' ' && currWord == ""){
            currChar++;
            continue;
        }
        if(line[currChar] == ' '){
            if(!isChild)
                this->addNext(currWord);
            else
                this->addChild(currWord);
            currWord = "";
            isChild = false;
            currChar++;
            continue;
        }
        if(line[currChar] == '('){
            isChild = true;
            currChar++;
            continue;
        }
        currWord = currWord + line[currChar];
        currChar++;
    }
    this->addNext(currWord);
}

void hlist::writeList(ofstream &outFile){
    int brackets = 1;
    base* curr = this->first;
    cout << '(';
    outFile << '(';
    while(curr->next || curr->child){
        cout << curr->value;
        outFile << curr->value;
        cout << ' ';
        outFile << ' ';
        if(curr->child){
            cout << '(';
            outFile << '(';
            brackets++;
            curr = curr->child;
        }
        else
            curr = curr->next;
    }
    cout << curr->value;
    outFile << curr->value;
    while(brackets){
        cout << ')';
        outFile << ')';
        brackets--;
    }
}

bool hlist::accFinder(base *first, base *second){
    base *temp = second;
```

```
    while(first){
        second = temp;
        while(second){
            if(first->value == second->value)
                return true;
            second = second->next;
        }
        first = first->next;
    }
    return false;
}

void hlist::repeatCheck(base * start){
    while(start){
        base* curr = start->next;
        while(curr){
            if(start->value == curr->value){
                if(curr->next){
                    curr->next->prev = curr->prev;
                    curr->prev->next = curr->next;
                }
                if(curr->child){
                    curr->child->parent = curr->prev;
                    curr->prev->next = nullptr;
                    curr->prev->child = curr->child;
                }
                if(!curr->next && !curr->child){
                    last = curr->prev;
                    last->next = nullptr;
                }
                base* temp = curr;
                curr = curr->prev;
                delete temp;
            }
            curr = curr->next;
        }
        start = start->next;
    }
}

base* hlist::notCheck(base *child, base *parent){
    if(parent->value == "OR" || parent->value == "AND"){
        bool result = accFinder(parent->next, child->next);
        if(!result)
            return parent;
        else{
            if(parent->parent->value == ""){
                first = parent->parent;
                last = first;
                if(parent->value == "OR")
                    first->value = "TRUE";
                else
                    first->value = "FALSE";
                deleteFrom(first->child);
                first->child = nullptr;
                return first;
            }
            else{
```

```
                last = parent->parent;
                if(parent->value == "OR")
                    addNext("TRUE");
                else
                    addNext("FALSE");
                deleteFrom(parent);
                base* curr = last;
                while(curr->prev)
                    curr = curr->prev;
                return curr;
            }
        }
    }
    if(parent->value == "NOT"){
        if(parent->parent->value == ""){
            first = parent->parent;
            last = first;
        }
        else
            last = parent->parent;
        base* curr = child->next;
        last->child = nullptr;
        while(curr){
            addNext(curr->value);
            curr = curr->next;
        }
        last->child = parent;
        parent->parent = last;
        base* temp = parent;
        while(temp->next)
            temp = temp->next;
        last = temp;
        deleteFrom(last->child);
        last->child = nullptr;
        return parent;
    }
    return parent;
}

base* hlist::andOrCheck(base *child, base *parent){
    if((child->value == "AND" && parent->value == "AND") || (child->value
== "OR" && parent->value == "OR")){
        child->parent->next = child->next;
        child->next->prev = child->parent;
        child->parent->child = nullptr;
        delete child;
    }
    return parent;
}

bool hlist::trueFalseCheck(base *curr){
    base* oper = curr;
    while(oper->prev)
        oper = oper->prev;
    if(oper == curr)
        return false;
    else{
```

```
        if((curr->value == "TRUE" && oper->value == "OR") || (curr->value
== "FALSE" && oper->value == "AND")){
            if(oper == first)
                first = first->parent;
            last = oper->parent;
            last->child = nullptr;
            addNext(curr->value);
            deleteFrom(oper);
            return true;
        }
        if(oper->value == "NOT"){
            if(last == curr){
                last = curr->prev;
                last->next = nullptr;
                curr->prev = nullptr;
            }
            else{
                if(curr->next){
                    curr->prev->next = curr->next;
                    curr->next->prev = curr->prev;
                    curr->next = nullptr;
                    curr->prev = nullptr;
                }
                else{
                    curr->prev->next = nullptr;
                    curr->prev->child = curr->child;
                    curr->child->parent = curr->prev;
                    curr->prev = nullptr;
                    curr->child = nullptr;
                }
            }
            if(oper == first){
                delete first->parent;
                first->parent = curr;
                curr->child = first;
                first = curr;
            }
            else{
                oper->parent->child = nullptr;
                oper->parent->next = curr;
                curr->prev = oper->parent;
                curr->child = oper;
                oper->parent = curr;
            }
        }
        return false;
    }
}

void hlist::simplifyList(){
    base* curr = first;
    while(curr){
        repeatCheck(curr);
        while(curr->next)
            curr = curr->next;
        curr = curr->child;
    }
    curr = last;
```

```
    while(curr->prev)
        curr = curr->prev;
    while(curr != first){
        if(curr->value == "NOT"){
            base *prevCurr = curr->parent;
            while(prevCurr->prev)
                prevCurr = prevCurr->prev;
            curr = notCheck(curr, prevCurr);
            continue;
        }
        if(curr->value == "AND" || curr->value == "OR"){
            base *prevCurr = curr->parent;
            while(prevCurr->prev)
                prevCurr = prevCurr->prev;
            curr = andOrCheck(curr, prevCurr);
            continue;
        }
        else{
            curr = curr->parent;
            while(curr->prev)
                curr = curr->prev;
        }
    }
    curr = first;
    while(curr){
        if(curr->value == "TRUE" || curr->value == "FALSE"){
            base* temp;
            if(curr->next)
                temp = curr->next;
            else
                temp = curr->child;
            bool answer = trueFalseCheck(curr);
            if(answer)
                break;
            curr = temp;
        }
        else{
            if(curr->next)
                curr = curr->next;
            else
                curr = curr->child;
        }
    }
    curr = first;
    while(curr){
        repeatCheck(curr);
        while(curr->next)
            curr = curr->next;
        curr = curr->child;
    }
}

int main()
{
    string file_name;
    cout << "Enter the name of an input file: " << endl;
    cin >> file_name;
    ifstream inFile(file_name);
```

```
    if (!inFile.is_open()){
        cout << "Permission denied or wrong path";
        return 0;
    }
    cout << "Result was saved in result.txt" << endl;
    ofstream outFile("/home/rostislav/result.txt");
    string line;
    while(getline(inFile, line)){
        if(line.length() == 0)
            continue;
        hlist* temp = new hlist;
        cout << line << " -> ";
        temp->readList(line);
        temp->simplifyList();
        temp->writeList(outFile);
        cout << endl;
        outFile << endl;
        delete temp;
    }
    cout << "End of work" << endl;
    inFile.close();
    outFile.close();
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 — Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарий
1.			Программа работает корректно
2.	(OR a a a) (TRUE)	(OR a) (TRUE)	Программа работает корректно
3.	(OR d (NOT d)) (AND d (NOT d)) (OR s (NOT b (NOT a))) (OR d (OR d (NOT a)))	(TRUE) (FALSE) (OR s a (NOT b)) (OR d (NOT a))	Программа работает корректно
4.	(NOT a (NOT FALSE)) (OR TRUE (AND a)) (AND q (AND FALSE (OR d)))	(FALSE (NOT a)) (TRUE) (FALSE)	Программа работает корректно