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: Иерархический список

Студентка гр. 7383 _____ Ханова Ю.А.

Преподаватель _____ Размочаева Н.В

Санкт-Петербург
2018

Содержание

Цель работы

Познакомиться с понятием иерархических списков и использованием их для работы с арифметическим выражением, получить навыки реализации иерархических списков на языке программирования C++.

Формулировка задачи: арифметическое, упрощение, проверка деления на 0, префиксная форма.

Реализация задачи

В данной лабораторной работе были реализованы структуры List(содержит операцию и флаг) и Pair(содержит указатели на голову и хвост элемента), а также следующие функции для работы с арифметическим выражением:

List* set_list(istream& is_str, List* hd_p=new List); - создает список, записывая операции и переменные из входной строки.

void out_list(List* head); - вывод списка;

void mod_list(List* head, List* prev); - преобразовывает выражения в соответствии с условием задания. Проверяется наличие рядом операции умножения или деления и символа 1 или сложение и вычитания рядом с 0, осуществляется проверка деления на 0;

bool isAtom(List* s); - проверяет, является ли список атомом;

List* get_head(List* s); - возвращает голову списка;

List* get_tail(List* s); - возвращает хвост списка;

bool isNull(List* s); - проверяет список на отсутствие элементов;

void destroy(List* s); - удаление списка;

List* make_pair(List* head, List* tail=nullptr); - создает структуру с парой указателей на голову и хвост;

List* make_atom(base op, base at); - создает атом с операцией;

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Выводы

В ходе выполнения данной лабораторной работы были освоены основные принципы работы с иерархическими списками на языке программирования C++. Также был создан набор функций для выполнения поставленной задачи.

ПРИЛОЖЕНИЕ А. Код программы

STRUCT.H

```
#ifndef STRUCT_H
#define STRUCT_H

typedef char base;

struct Pair
{
    struct List* hd;
    struct List* tl;
};

struct List
{
    bool flag; //true: atom, false: pair
    base operation; //if Pair - \0
    union {
        base atom;
        struct Pair p_ptr;
    }node{};
};

bool isAtom(List* s)
{
    if(!s)
        return false;
    else
        return s->flag;
}

List* get_head(List* s)
{
    if(s!=nullptr)
    {
        if(!isAtom(s))
            return s->node.p_ptr.hd;
        else
            throw "Error: Head(atom)";
    }
    else
        throw "Error: Head(nil)";
}

List* get_tail(List* s)
```

```
{
    if(s!=nullptr)
    {
        if(!isAtom(s))
            return s->node.p_ptr.tl;
        else
            throw "Error: Tail(atom)";
    }
    else
        throw "Error: Tail(nil)";
}

bool isNull(List* s)
{
    return s==nullptr;
}

void destroy(List* s)
{
    if (!s)
    {
        if (!isAtom(s))
        {
            destroy(s->node.p_ptr.hd);
            destroy(s->node.p_ptr.tl);
        }
        delete s;
    }
}

List* make_pair(List* head, List* tail=nullptr)
{
    if(isAtom(tail))
        throw "Error: Tail(nil).";
    List* p = new List;
    if(!p)
        throw "Memory is not enought.";
    p->flag = false;
    p->node.p_ptr.hd = head;
    p->node.p_ptr.tl = tail;

    return p;
}

List* make_atom(base op, base at)
{
    List* p = new List;
    if(!p)
        throw "Memory is not enought.";
    p->flag = true;
    p->operation = op;
    p->node.atom = at;
```

```
    return p;
}

#endif // STRUCT_H
```

FUNCTIONS.H

```
#include <iostream>
#include <cctype>
#include "struct.h"

using namespace std;

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

List* set_list(istream& is_str, List* hd_p=new List)
{
    char cur_ch;
    if(is_str>>cur_ch)
    {
        if(cur_ch=='+' || cur_ch=='-' || cur_ch=='*' || cur_ch=='/')
        {
            hd_p = make_pair(make_atom(cur_ch, '\0'),
make_pair(set_list(is_str)));
            if(is_str>>cur_ch)
                get_head(hd_p)->node.atom = cur_ch;
            else
                throw "Operations >= operators.";
        }
        else if(isalnum(cur_ch))
        {
            hd_p = make_pair(make_atom('\0', cur_ch));
            return hd_p;
        }
        else
            throw "Invalid input.";
    }

    return hd_p;
}

void out_list(List* head)
{
    if(head->node.p_ptr.tl)
    {
        out_list(get_head(get_tail(head)));
```

```
    }
    if(get_head(head)->operation)
        cout << get_head(head)->operation;
    cout << get_head(head)->node.atom;
}

void mod_list(List* head, List* prev)
{
    if(head->node.p_ptr.tl)
    {
        mod_list(get_head(get_tail(head)), head);
    }
    else
    {
        if((get_head(head)->node.atom=='0' &&(get_head(prev)-
>operation=='+' || get_head(prev)->operation=='-'))
           || (get_head(head)->node.atom=='1' && get_head(prev)-
>operation=='*'))
        {
            get_head(prev)->operation = '\0';
            prev->node.p_ptr.tl = nullptr;
            destroy(head);
            return;
        }
    }

    if(prev==nullptr)
    {

    }

    if(head->node.p_ptr.hd->node.atom=='0' && head->node.p_ptr.hd-
>operation=='/')
        throw "Delenie na 0.";
    else if((get_head(head)->node.atom=='0' && (get_head(head)-
>operation=='+' || get_head(head)->operation=='-'))
      || (get_head(head)->node.atom=='1' && (get_head(head)-
>operation=='*' || get_head(head)->operation=='/')))
    {
        if(prev==nullptr)
        {
            auto temp = head;
            head = get_tail(head)->node.p_ptr.hd;
            destroy(temp);
        }
        else
        {
            get_tail(prev)->node.p_ptr.hd=get_head(get_tail(head));
            destroy(head);
            return;
        }
    }
```

```
}


#endif // FUNCTIONS_H
```

MAIN.CPP

```
#include <fstream>
#include <exception>
#include <sstream>
#include "functions.h"
#include "struct.h"

using namespace std;

int main()
{
    string temp_str;
    stringbuf str_buf;
    istream is_str(&str_buf);
    string file_name;
    filebuf file;
    int t = 3;

    while(true)
    {
        try
        {
            List* head;

            char istr_null ='\0';

            switch(t)
            {
            case 1:
                cout << "Enter a name of the data-file:" << endl;
                getline(cin, file_name);

                if(!file.open(file_name, ios::in))
                    throw "Input file isn't opened.";
                else
                {
                    auto size = file.in_avail();
                    if(!size)
```

```
            throw "File is empty.";

        char temp_ch;

        cout << "File contains: ";
        for(auto c_size = 0; c_size<size; c_size++)
        {
            temp_ch=file.sbumpc();
            cout << temp_ch;
            if(temp_ch != ' ' && temp_ch != '\n')
                temp_str.append(1, temp_ch);
        }
        cout << endl;

        file.close();
        t = -1;
    }
    break;
case 2:
    cout << "Enter an expression." << endl;
    getline(cin, temp_str);
    t = -1;
    break;
case -1:
    str_buf.str(temp_str);
    head = set_list(is_str);
    out_list(head);
    cout << endl;
    mod_list(head, nullptr);
    out_list(head);
    cout << endl;
    if(is_str>>istr_null)
        throw "Operations < operators.";

    temp_str.clear();
    is_str.clear();
    t = 3;

    break;
case 0:
    return 0;
default:
    cout << "Enter \"1\" to input data from file." << endl
         << "Enter \"2\" to input data from console." << endl
         << "Enter \"0\" to exit."  << endl;
    string t_str;
    getline(cin, t_str);
    t = stoi(t_str);
    if(t == -1)
        t = 3;
    break;
}
```

```
        }
        catch (char const* err)
        {
            cout << err << endl;
            t = 3;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б. Тестовые случаи

Таблица 1 - Результаты тестов.

input	output	True/false
+-AB0	A-B	True
+-*A1BC	C+B-A	True
+*/-AB0CD	Delenie na 0!	True
-+*/A1B0C	A*B-C	True