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

Студентка гр. 7383 _____ Маркова А. В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Содержание

Цель работы

Ознакомиться с основными понятиями иерархического списка и научиться реализовывать его на языке программирования C++.

Формулировка задачи: вычислить глубину (число уровней вложения) иерархического списка как максимальное число одновременно открытых левых скобок в сокращенной скобочной записи списка; принять, что глубина пустого списка и глубина атомарного S – выражения равны нулю; например, глубина списка (a (b () c) d) равна двум.

Реализация задачи

В данной работе используется главная функция `main()` и дополнительные функции `int max (int a, int b)`, `int listDeepth (lisp list)`, структуры `s_expr` и `two_ptr`, а также базовые функции для создания иерархического списка.

После запуска программы функция `main()` выводит меню на консоль, где можно выбрать пункт, соответствующий нужной операции. Считывается целое число и при помощи оператора `switch()`, выполняется необходимое действие. При нажатие «1», программа считает глубину вложенности списка, считанного с файла, при нажатие «2» пользователь сам вводит нужный иерархический список для проверки. При выборе «3» функция завершает свою работу. Если было введено другое значение, отличное от стандартных, то программа выведет сообщение об ошибке: «Некорректный выбор!» и завершит работу. Программа завершается при выборе «3» или введение неизвестной команды, в противном случае – ожидает дальнейших указаний.

Для реализации иерархического списка использовались две структуры: `struct s_expr` и `struct two_ptr`. Структура `struct two_ptr` содержит в себе два указателя `s_expr *hd` и `s_expr *tl`. Структура `struct s_expr` содержит переменную `bool tag`, которая в зависимости от того, является элемент атомом или подсписком присваивает значение `true` и `false` соответственно. Также эта структура содержит объединение двух типов, `base atom` и `two_ptr pair`.

Функции-селекторы: `head` и `tail`, выделяющие «голову» и «хвост» списка соответственно. Если «голова» списка не атом, то функция `head`, возвращает список, на который указывает голова пары, т.е. подсписок, находящийся на следующем уровне иерархии. Если же «голова» списка -

атом, то выводится сообщение об ошибке и функция прекращает работу.

Функция `tail` работает аналогично функции `head`, но только для «хвоста».

Функции-конструкторы: `cons`, создающая точечную пару (новый список из «головы» и «хвоста»), и `make_atom`, создающая атомарное выражение. При создании нового выражения требуется выделение памяти. Если памяти нет, то `p==NULL` и это приводит к выводу соответствующего сообщения об ошибке. Если «хвост» - не атом, то для его присоединения к «голове» требуется создать новый узел (элемент), головная ссылка которого будет ссылкой на «голову» этого «хвоста», а хвостовая часть элемента - ссылкой на его «хвост».

Функции-предикаты: `isNull`, проверяющая список на отсутствие в нем элементов, и `isAtom`, проверяющая, является ли список атомом. Если элемент - атом, тогда функция возвращает значение `tag`, которое равно `true`, и значение `false`, если «голова-хвост». В случае пустого списка значение предиката `false`.

Функции-деструкторы: `delete` и `destroy`. Функция `delete` удаляет текущий элемент из списка, а функция `destroy` удаляет весь список путем вызова функции `delete` и вызова самой себя.

Функция `getAtom` возвращает нам значение атома.

Функция `copy_lisp` - функция копирования списка.

Функция `listDeepth` - считает количество уровней вложенности иерархического списка по открытие левой скобки.

Функция `max` - сравнивает значение счетчика глубины и выбирает наибольший.

Разберем для примера работы программы строку (v (d) c):

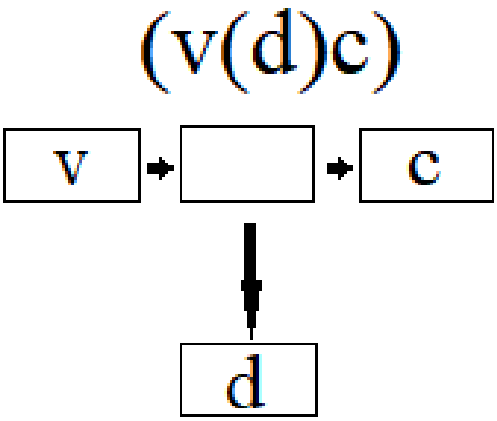

Рисунок 1 – пример работы программы

Сначала программа заполнит элемент с «v», затем встретив левую скобку она создаст пустой элемент, который будет содержать указатель на следующий уровень вложенности, вызов будет являться рекурсивным. После того как элемент «d» будет создан, программа продолжит обрабатывать строку и когда считает правую скобку, то поднимется на уровень выше и продолжит работу с уровнем, пока не закончится строка. Глубина вложенности в данном примере равна двум.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04, с использованием компилятора g++ версии 5.4.0 20160609. В других ОС и компиляторах тестирование не проводилось.

Программа может быть скомпилирована с помощью команды:

g++ - Wall <имя файла>.c

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно заметить, что во втором тесте программа ведет себя неверно, поэтому была исправлена программа: добавлено в `default` изменение значения флага на `false`, чтобы функция не зацикливалась.

Так же была выявлена ошибка в пятом тесте, в связи с этим в программу добавилась строка, которая обнуляла значения счетчика.

После, тестовые случаи не выявили неправильного поведения программы, что говорит о том, что по результатам тестирование было показано, поставленная задача была выполнена.

Выводы

В ходе лабораторной работы были изучены основные свойства и понятия иерархического списка. Была написана программа реализующая рекурсивную функцию, а так же создана программа для подсчета максимальной глубины вложенности на языке программирования C++.

ПРИЛОЖЕНИЕ А

Тестовые случаи

Ввод	Вывод	Верно?
1 В файле: «( )»	Список пуст!	Да
4	Некорректный выбор! Некорректный выбор! …… Некорректный выбор!	Нет
8	Некорректный выбор! До свидания!	Да
1 (файл не был создан)	Файл не может быть открыт!	Да
2 (a ( )) 2 ( ) 2 (d f() d(d))	Максимальный уровень вложенности 1 Максимальный уровень вложенности 1 Максимальный уровень вложенности 1	Нет
2 (F f() f(d(d))) 3	Максимальный уровень вложенности 3 До свидания!	Да
2 (D s)	Максимальный уровень вложенности 1	Да

ПРИЛОЖЕНИЕ Б

Код программы

```
#include"list.h"

#define N 50

using namespace h_list;

int main()
{
    FILE* f;
    stringbuf exp;//входная последовательность символов
    int run = 1, pick;
    string list;
    int count;
    lisp l1;
    cout << "\033[34m\tЗдравствуйте!Выберите что вы хотите:\033[0m\n 1) Нажмите 1,
чтобы считать с файла.\n 2) Нажмите 2, чтобы считать с консоли.\n 3) Нажмите 3,
чтобы выйти из программы.\n" << endl;
    while (run) {
        cin >> pick;
        cin.ignore();
        switch(pick) {
        case (1): {
            ifstream infile("test.txt");
            if(!infile)cout<<"Входной файл не может быть открыт!"<<endl;
            else {
                read_lisp(l1, infile);
                cout << "Введённый список:" << endl;
                write_lisp(l1, cout);
                cout << endl;
                count = listDeepth(l1);
                cout << "Максимальный уровень вложенности: " << count << endl;
            }
            break;
        }
        case (2): {
            cout << "введите строку:" << endl;
            getline(cin, list);
            istream is_str(&exp);//проверка на строку
            exp.str(list);
            read_lisp(l1, is_str);
            count = listDeepth(l1);
            cout << "Максимальный уровень вложенности: " << count << endl;
            break;
        }
        case (3): {
            cout<<"\033[34m До свидания!\033[0m"<<endl;
```

```
            run=false;
            break;
        }
        default: {
            cout<<"\033[31m НЕВЕРНЫЙ ВВОД!\033[0m"<<endl;
            break;
        }
        }
    }
}

#include "list.h"

using namespace std;

namespace h_list //используемое пространство имен
{

//.................................
lisp head(const lisp s) //создание головы списка
{
    if (s != NULL)
        if (!isAtom(s))return s->node.pair.hd;//в начале узел идёт
        else {
            cerr << "\033[31m Error: Head(atom) \n\033[0m";
            exit(1);
        }//cerr - для вывода ошибок
    else {
        cerr << "\033[31m Error: Head(nil) \n\033[0m";
        exit(1);
    }
}
//.................................
bool isAtom(const lisp s)//проверка на атом
{
    if (s == NULL) return false;//s - указатель на элемент списка
    else return (s->tag);//false - узел, true - атом
}
//.................................
bool isNull(const lisp s)//проверка на пустоту
{
    return s == NULL;
}
//.................................
lisp tail(const lisp s)
{
    if (s != NULL) if (!isAtom(s))  return s->node.pair.tl;//в конце узел идёт
        else {
            cerr << "\033[31m Error: Tail(atom) \n\033[0m";
            exit(1);
        }
```

```
    else {
        cerr << "\033[31m Error: Tail(nil) \n\033[0m";
        exit(1);
    }
}
//.....................................
lisp cons(const lisp h, const lisp t)//создает новый список из головы и хвоста
{
    lisp p;
    if (isAtom(t)) {
        cerr << "\033[31m Error: Cons(*, atom) \n\033[0m";
        exit(1);
    }
    else {
        p = new s_expr;//создание узла
        if (p == NULL) {
            cerr << "\033[31m Memory not enough \n\033[0m";
            exit(1);
        }
        else
        {
            p->tag = false;
            p->node.pair.hd = h;
            p->node.pair.tl = t;
            return p;
        }
    }
}
//.........................
lisp make_atom(const base x)//создание атомарного выражения
{
    lisp s;
    s = new s_expr;//оператор, обеспечивающий выделение динамической памяти
    s->tag = true;
    s->node.atom = x;//заполнение элемента списка
    return s;
}

//.........................
void destroy(lisp s)//удаляет весь списо
{
    if (s != NULL) {
        if (!isAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;//оператор, возвращающий память, выделенную new, обратно в кучу
    };
    s = NULL;
}
//.........................
```

```
base getAtom(const lisp s)//возвращает значение атома
{
    if (!isAtom(s)) {
        cerr << "\033[31m Error: getAtom(s) for !isAtom(s) \n\033[0m";
        exit(1);
    }
    else return (s->node.atom);
}

//.........................
// ввод списка с консоли
void read_lisp(lisp& y, istream& in)//элемент, атом
{
    base x;
    do in >> x;
    while (x == ' ');
    read_s_expr(x, y, in);
}
//.........................
void read_s_expr(base prev, lisp& y, istream& in)
{   //prev   — ранее прочитанный символ}
    if (prev == ')') {
        cerr << "\033[31m ! List.Error(1) \033[0m" << endl; //отсутствует
октрывающая скобка
        exit(1);
    }
    else if (prev != '(') y = make_atom(prev);
    else read_seq(y, in);
}
//.........................
void read_seq(lisp& y, istream& in)//создание списка
{
    base x;
    lisp p1, p2;

    if (!(in >> x)) {
        cerr << "\033[31m ! List.Error(2) \033[0m" << endl; //отсутствует
закрывающая скобка
        exit(1);
    }
    else {
        while (x == ' ') in >> x;
        if (x == ')') y = NULL;
        else {
            read_s_expr(x, p1, in);
            read_seq(p2, in);
            y = cons(p1, p2);
        }
    }
}
//.........................
```

```
void write_lisp(const lisp x, ostream& out)
{   //пустой список выводится как ()
    if (isNull(x)) out << " ()";
    else if (isAtom(x)) out << ' ' << x->node.atom;
    else  { //непустой список}
        out << " (";
        write_seq(x, out);
        out << " )";
    }
}
//.........................
void write_seq(const lisp x, ostream& out)
{   //выводит последовательность элементов списка без обрамляющих его скобок
    if (!isNull(x)) {
        write_lisp(head(x), out);
        write_seq(tail(x), out);
    }
}
//.........................
lisp copy_lisp(const lisp x)//функция копирования списка
{
    if (isNull(x)) return NULL;
    else if (isAtom(x)) return make_atom(x->node.atom);
    else return cons(copy_lisp(head(x)), copy_lisp(tail(x)));
}
//.........................
int max(int a, int b)
{
    if (a > b) return a;
    return b;
}
//.........................
int listDeepth(lisp list)
{
    if (isNull(list)) return 0;
    if (isAtom(list)) return -1;
    return max(listDeepth(head(list)) + 1, listDeepth(tail(list)));
}

} // end of namespace h_list
```