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 _____ Прокопенко Н.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

Содержание

Цель работы

Познакомиться с основными понятиями и приемами работы с иерархическими списками, получить навыки программирования рекурсивных функций на языке программирования C++.

Формулировка задачи: проверить структурную идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих (под)списках) одинаково, при этом атомы могут отличаться);

Реализация задачи

В данной работе было написано несколько функций для реализации задачи. Перечень функций:

bool Check(istringstream &s) - функция для проверки скобок. В данную функцию передается поток. В данной функции проверяется соответствие открывающих и закрывающих скобок. Функция возвращает значение true или false.

string delet(string str) – функция удаления пробелов. Данной функции передается строка, которая проверяется на пробелы. При нахождении данный пробел удаляется. После чего измененная строка возвращается.

bool isAtom(const lisp s) - функция, проверяющая по тегу является ли элемент атомом. Функция возвращает значение tag, которое равно True, если элемент ─ атом, и значение False, если нет. В случае пустого списка значение предиката False.

bool isNull(const lisp s) - функция для проверки, является ли список пустым.

lisp head(const lisp s) - функция head, выделяющая «голову» списка. Если «голова» списка не атом, то функция head возвращает список, на который указывает голова пары, т.е. подсписок, находящийся на следующем уровне иерархии.

bool isEqual_lisp(const lisp f, const lisp s) - функция проверки структурной идентичности двух иерархических списков. Если оба списка пусты, то возвращается true. Если хотя бы у одного списка соответствующий элемент не является атом, то функция возвращает false. Если у обоих списком соответствующий элемент является атом, то функция возвращает true. Если у обоих списков соответствующий элемент не является атомом, то вызывается функция bool isEqual_seg(const lisp f, const lisp s).

Были созданы функции ввода иерархического списка. Для ввода иерархического списка, представленного сокращенной скобочной записью, используется функция read_lisp. Эта функция использует внутри себя обращение к функции read_s_expr, а она, в свою очередь, обращение к read_seq.

lisp cons(const lisp h, const lisp t) - функция cons. При создании нового S-выражения требуется выделение памяти. Если памяти нет, то p == NULL и это приводит к выводу 4 соответствующего сообщения об ошибке. Если «хвост» ─ не атом, то для его присоединения к «голове» требуется создать новый узел (элемент).

lisp tail(const lisp s) - создана функция tail, выделяющая «хвост» списка. Если «хвост» списка не атом, то функция tail возвращает список, на который указывает хвост пары.

int main ( ) - головная функция. В данной функции выводится меню программы, после чего считывается выбранный вариант и запускается функция switch( ), которая находясь в теле цикла while( ), будет выполняться до тех пор, пока пользователь не захочет выйти из программы.

При введении пользователем «1», программа запрашивает две входные строки, вводимые с клавиатуры. Затем вызываются функция проверки скобок и функция удаления пробелов, описанные выше. После чего вызываются функции создания иерархических списков и проверки их

на структурную идентичность, после чего выводится результат проверки и пользователю предлагается выбрать другой пункт меню.

При введении пользователем «2», программа предлагает ввести имя входного файла, и если файл с таким файлом существует, то оттуда считываются строки, которые обрабатываются. Вызываются функции создания иерархических списков и проверки их на структурную идентичность. Если же такого файла не существует, то выводится соответствующая надпись и предлагается повторить выбор.

При введении пользователем «0», происходит выход из программы.

При введении пользователем других данных предлагается выбрать один из пунктов меню, описанных выше. Код программы представлен в Приложении А.

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 лабораторной работы были получены навыки работы с иерархическими списками. Иерархические списки позволяют упростить написание и повысить читабельность кода, если требуется выполнить задачу, подразумевающую работу с повторяющимися типами данных. Рекурсия, использованная при реализации списков, позволяет облегчить выполнение поставленной задачи.

ПРИЛОЖЕНИЕ А. Код программы

```
#include <string>
#include <iostream>
#include <cstdlib>
#include <sstream>
#include <fstream>
#include <cstring>
#include <cstdio>

using namespace std;

typedef char* base;      // базовый тип элементов (атомов)

struct s_expr {
    bool tag; // true: atom, false: pair
    union {
        base atom;
        struct two_ptr {

            s_expr *hd;
            s_expr *tl;
        } pair;
    } node;
};

typedef s_expr *lisp;

lisp head(const lisp s);
lisp tail(const lisp s);
lisp cons(const lisp h, const lisp t);
lisp make_atom(const base x);
bool isAtom(const lisp s);
bool isNull(const lisp s);
void destroy(lisp s);
string lisp_to_string(const lisp x);
string seq_to_string(const lisp x);
void read_lisp(lisp &y, istringstream &ss);
void read_s_expr(char prev, lisp &y, istringstream &ss);
void read_seq(lisp& y, istringstream &ss);
bool isEqual_lisp(const lisp f, const lisp s);
bool isEqual_seg(const lisp f, const lisp s);

lisp make_atom(const base x) {

    lisp s;
    s = new s_expr;
```

```
    s->tag = true;
    s->node.atom = x;
    return s;
}

bool isAtom(const lisp s) {

    if (s == NULL) {
        return false;
    } else {
        return (s->tag);
    }
}

bool isNull(const lisp s) {

    return s == NULL;
}


lisp head(const lisp s) { // PreCondition: not null (s)

    if (s != NULL) {
        if (!isAtom(s)) {
            return s->node.pair.hd;
        }
        else {
            cerr << "Error: Head(atom) \n";
            exit(1);
        }
    } else {
        cerr << "Error: Head(nil) \n";
        exit(1);
    }
}

lisp tail(const lisp s) {// PreCondition: not null (s)

    if (s != NULL) {
        if (!isAtom(s)) {
            return s->node.pair.tl;
        } else {
            exit(1);
        }
    } else {
        exit(1);
    }
```

```
}

lisp cons(const lisp h, const lisp t) { // PreCondition: not isAtom
(t)

    lisp p;
    if (isAtom(t)) {
        exit(1);
    } else {
        p = new s_expr;
        if (p == NULL) {
            exit(1);
        } else {
            p->tag = false;
            p->node.pair.hd = h;
            p->node.pair.tl = t;
            return p;
        }
    }
}

void destroy(lisp s) {

    if (s != NULL) {
        if (!isAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    }
}

void read_lisp(lisp& y, istringstream &ss) {

    char x;
    ss>>x;
    read_s_expr(x, y, ss);
}

void read_s_expr(char prev, lisp& y, istringstream &ss) {

    int i = 0;
    char ch = prev;
    streampos oldpos;
    base str = (base)calloc(30, sizeof(char));
    if (prev == ')') {
      exit(1);
```

```
    } else if (prev != '(') {
        while ( isalpha(ch) && ch != ')') {
            str[i] = ch;
            i++;
            oldpos = ss.tellg();
            ch=ss.get();
            while (ch == ' ') {
                ch=ss.get();
            }
        }
        if (ch == ')') {
           ss.seekg(oldpos);
        }
        y = make_atom(str);
    } else {
        read_seq(y, ss);
    }
}



void read_seq(lisp& y, istringstream &ss) {

    char x;
    lisp p1, p2;
    if (!(ss >> x)) {
        exit(1);
    } else {
        if (x == ')') {
            y = NULL;
        } else {
                read_s_expr(x, p1, ss);
                read_seq(p2, ss);
                y = cons(p1, p2);
        }
    }
}



string lisp_to_string(const lisp x) {//пустой список выводится как
()

    string str;
    if (isNull(x)) {
        str += "()";
    }
```

```
    else if (isAtom(x)) {
        str += x->node.atom;
        str += " ";
    } else { //непустой список}
        str +=  "( ";
        str += seq_to_string(x);
        str += ") ";
    }
    return str;
}


string seq_to_string(const lisp x) {//выводит последовательность
элементов списка без обрамляющих его скобок
    if (!isNull(x)) {
        return lisp_to_string(head(x)) + seq_to_string(tail(x));
    }
    return "";
}

base getAtom(const lisp s) {

    if (!isAtom(s)) {
        exit(1);
    } else {
        return (s->node.atom);
    }
}

bool isEqual_lisp(const lisp f, const lisp s) {

    if (isNull(f) && isNull(s)) {
        return true;
    } else if ((!isAtom(f) && isAtom(s)) || (isAtom(f) &&
!isAtom(s)))
      return false;
     else if(!isAtom(f) && !isAtom(s)) {
        return isEqual_seg(f, s);
    }else if(isAtom(f) && isAtom(s))
      return true;
    else return false;
}

bool isEqual_seg(const lisp f, const lisp s) {

    if (!isNull(f) && !isNull(s)) {
```

```
        return isEqual_lisp(head(f), head (s)) &&
isEqual_seg(tail(f), tail(s));
    }
    return isNull(f) && isNull(s);
}

int print(bool result, int sw_var){
  if(result)
      cout << "True. These are structurally identical lists.\n";
  else
      cout << "False. These are not structurally identical
lists.\n";
  return sw_var = 4;
}

bool Check(istringstream &s) {
      char x;
      int countopen=0;
      int countclose=0;
      while (s >> x) {
            if (x == '(')
                  countopen++;
            if (x == ')')
                  countclose++;
      }
      if (countopen == countclose)
            return true;
      else
            return false;
}

string delet(string str){
  int i=0;
  while(str.length()>i){
    if(str[i] == ' ')
      str.erase(i,1);
    else  i++;
  }
  return str;
}

int main() {

      lisp* s = new lisp;
      lisp* f = new lisp;
      string str1, str2;
      int sw_var;
```

```
string file_name;
fstream file;
cout << "Menu" << '\n';
cout << "0-exit from the program" << '\n';
cout << "1-input a string from the keyboard" << '\n';
cout << "2-input a line from a file" << '\n';
cin >> sw_var;
cin.ignore();
while (sw_var)
{
      switch (sw_var)
      {
      case 1: {
            cout << "Enter the first line :" << '\n';
            getline(cin, str1);
 istringstream ss1(delet(str1));
            if (!Check(ss1)) {
                  cout << "Error\n";
   sw_var=4;
                  break;
            }
            read_lisp(*f, ss1);
            cout << "Enter the second line :" << '\n';
            getline(cin, str2);
            istringstream ss2(delet(str2));
            if (!Check(ss2)) {
                  cout << "Error\n";
   sw_var=4;
                  break;
            }
            read_lisp(*s, ss2);
            sw_var = print(isEqual_lisp(*f, *s), sw_var); }
                  break;
      case 2: {
            cout << "Enter the name of the file:" << '\n';
            cin >> file_name;
            cin.ignore();
            file.open(file_name, fstream::in);
            if (!file.is_open()) {
                  cout << "Error opening file.\n";
                  sw_var = 4;
            }
            else {
                  getline(file, str1);
                  istringstream ss1(delet(str1));
                  read_lisp(*f, ss1);
                  getline(file, str2);
```

```
                    istringstream ss2(delet(str2));
                    read_lisp(*s, ss2);
                    sw_var = print(isEqual_lisp(*f, *s), sw_var);
                    file.close();
               }}
                    break;
          case 0:
               break;
          default: {
               cout << "Enter again.\n";
               cin >> sw_var;
               cin.ignore(); }
                     break;
          }
     }
     return 0;
}
```

ПРИЛОЖЕНИЕ Б. Тестовые случаи

Таблица 1 - Результаты тестов.

Input	Output	True/False
(HHH) (j)	True. These are structurally identical lists.	True
(DFG(HJ)) (VGB(BH(MK)))	False. These are not structurally identical lists.	False
(Q) (МП(O))	False. This is not the concept of brackets.	False
(TO(P(P))) (OFG(JK))	False. These are not structurally identical lists.	True
(OFG(JK)) (VF(N))	True. These are structurally identical lists.	True
(BJJM))))) (BHN)	Error	True