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

Студент гр. 9303 _____ Дюков В. А.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург
2020

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки.

Задание.

Написать рекурсивную функцию или процедуру, формирующую линейный список номеров всех вхождений одного бинарного коромысла в другое.

Основные теоретические положения.

В практических приложениях возникает необходимость работы с более сложными, чем линейные списки, нелинейными конструкциями. Рассмотрим одну из них, называемую иерархическим списком элементов базового типа El или S-выражением.

Определим соответствующий тип данных S_expr (El) рекурсивно,используя определение линейного списка (типа L_list):

< S_expr (El) > ::= < Atomic (El) > | < L_list (S_expr (El)) >
< Atomic (E) > ::= < El >
< L_list (El) > ::= < Null_list > | < Non_null_list(El) >
< Null_list > ::= Nil
< Non_null_list (El) > ::= < Pair(El) >
< Pair (El) > ::= (< Head_l(El) > . < Tail_l(El) >)
< Head_l (El) > ::= < El >
< Tail_l (El) > ::= < L_list (El) >

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.

Выполнение работы.

Для выполнения работы были реализованы следущие функции:

- void *list_comp*(const rocker_arm* data, const rocker_arm& temp, const std::string& path, list& result);
- void *list_print*(std::ostream& out, const list& data, const std::string& str);
- void *list_delete*(list& data);
- void *rocker_arm_del*(rocker_arm& data);
- void *arm_del*(arm& data);
- void *cargo_del*(cargo& data);
- void *menu_print*(bool hat, const std::string str = "", std::ofstream* ptr = nullptr);
- int *kb_react*(const char* first, const char* second);

Также были перегружены следующие операторы:

- bool *operator*==(const rocker_arm& first, const rocker_arm& second);
- bool *operator*==(const arm& first, const arm& second);
- bool *operator*==(const cargo& first, const cargo& second);
- std::istream& *operator*>>(std::istream& in, rocker_arm& data);
- std::istream& *operator*>>(std::istream& in, arm& data);
- std::istream& *operator*>>(std::istream& in, cargo& data);
- std::istream& *operator*>>(std::istream& in, s_string& data);

Перегруженные операторы сравнения *operator==* – являются рекурсивными и вызывают сами себя в определённой последовательности. Результат выполнения всей цепочки операторов определяется либо оператором сравнения типов *cargo*, либо типов *arm*. С помощью них можно установить равенство либо грузов, либо плеч, либо коромысел двух бинарных коромысел.

Перегруженные операторы бинарного сдвига *operator>>* – являются рекурсивными операторами считывания и вызывают сами себя в определённой последовательности. Каждый оператор считывает определённую структуру, создаёт новый объект соответвенного типа и расширяет иерархический список, после чего запускает считывание следующей структуры. В случае, если считывание произошло не верно, оператор удаляет все данные и объекты текущей итерации. Исключением является оператор считывания типа *s_string* , который предназначен для правильного считывание бинарного коромысла из файла.

Функция *list_comp* – рекурсивная функция, которая ищет и запоминает все вхождения переменной *temp* – малого бинарного коромысла в *data* - большое. Функция принимает адрес на большое и ссылку на маленькое бинарные коромысла, ссылку на строку пути до вхождения и неконстантную ссылку на на линейный список. В каждой итерации функция расширяет список, если выполнены условия равенства коромысел.

Функция *list_print* – функция, выводящая список из вхождений одного бинарного коромысла в другое.

Функция *list_delete* – функция, освобождающая память из под списка вхождений.

Функции *rocker_arm_del, arm_del, cargo_del* – рекурсивные функции и вызывают сами себя в определённой последовательности и последовательно освобождают память из каждой структуры коромысла.

В функции *main* реализован интерфейс взаимодействия с программой, а конкретно: считывание файла с исходными данными, перемещение между итерациями программы и выход из программы. Также функция производит запись каждой итерации программы в файл *result.txt* .

Исходный код программы приведён в Приложении А.

Тестирование

№ т.	Входные данные	Результат	Вывод
1	((1 2) (1 2)) ((1 2) (1 2))	1. Два бк равны -- ((1 2) (1 2))	Программа работает верно
2	((1 ((1 2) (1 2))) (2 ((1 2) (1 2)))) ((1 2) (1 2))	1. Корневое Коромысло -> Левое плечо -> || -> ((1 2) (1 2)) 2. Корневое Коромысло -> Правое плечо -> || -> ((1 2) (1 2))	Программа работает верно
3	((1 ((2 ((1 3) (4 2)))(3 3)))(4 ((2 2) (6 2)))) ((1 3) (4 2))	1. Корневое Коромысло -> Левое плечо -> Коромысло -> Левое плечо -> || -> ((1 3) (4 2))	Программа работает верно
4	((1 ((2 ((1 3) (4 2)))(3 3)))(4 ((1 3) (4 2)))) ((2 ((1 3) (4 2)))(3 3))	1. Корневое Коромысло -> Левое плечо -> || -> ((2 ((1 3) (4 2)))(3 3))	Программа работает верно
5	((a ((jreiuuons)()))) (beryvbu)	Ошибка ввода БинКор, в котором идёт поиск совпадений. Ошибка ввода БинКор, совпадения с которым в поиске.	Программа работает верно

Выводы

В ходе выполнения программы были изучены такие нелинейные конструкции, как бинарные коромысла, и способы их рекурсивной обработки.

В процессе выполнения работы были испытаны трудности при реализации вывода элементов списка вхождений (вывода пути до вхождений).

Для их решения пришлось проивить внимательность и переработать основную функцию *list_comp.*

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

header.h:

```
#pragma once

#include <iostream>
#include <fstream>
#include <sstream>
#include <Windows.h>
#include <conio.h>
#include <string>

struct rocker_arm;
struct cargo {

	int weight;
	rocker_arm* next_ra;
	cargo() : next_ra(nullptr), weight(-1) {}
};
struct arm {

	int lenght;
	cargo* next_c;
	arm() : next_c(nullptr), lenght(-1) {}
};
struct rocker_arm {

	arm* left_arm;
	arm* right_arm;
	rocker_arm() : left_arm(nullptr), right_arm(nullptr) {}
};
struct node {

	node* next;
	rocker_arm* data;
	std::string path;
```

```
};
struct s_string {

	std::string str;
};

typedef node* list;

enum flags {F_EXIT, F_START, F_CONSOL, F_FILE, F_RESULT_S, F_FRES, F_CRES};

bool operator==(const rocker_arm& first, const rocker_arm& second);
bool operator==(const arm& first, const arm& second);
bool operator==(const cargo& first, const cargo& second);

std::istream& operator>>(std::istream& in, rocker_arm& data);
std::istream& operator>>(std::istream& in, arm& data);
std::istream& operator>>(std::istream& in, cargo& data);

void rocker_arm_del(rocker_arm& data);
void arm_del(arm& data);
void cargo_del(cargo& data);
```

main.cpp:

```
#include "header.h"

// - - - - - main  recursion - - - - - //

bool operator==(const rocker_arm& first, const rocker_arm& second) {

	return ((*first.left_arm == *second.left_arm) && (*first.right_arm ==
*second.right_arm));
}

bool operator==(const arm& first, const arm& second) {

	return (*first.next_c == *second.next_c) && (first.lenght ==
second.lenght);
}

bool operator==(const cargo& first, const cargo& second) {
```

```
	if (first.next_ra && second.next_ra)
		return *first.next_ra == *second.next_ra;
	if (!first.next_ra && !second.next_ra)
		return (first.weight == second.weight);
	return false;
}

void list_comp(const rocker_arm* data, const rocker_arm& temp, const
std::string& path, list& result) {

	if (data && temp.left_arm && data->left_arm) {

		if (*data == temp)
			if (!result) {

				result = new node;
				result->data = (rocker_arm*)&data;
				result->path = path;
				result->next = nullptr;
			}
			else {

				node* head;
				for (head = result; head->next; head = head->next);
				head->next = new node;
				head->next->data = (rocker_arm*)&data;
				head->next->path = path;
				head->next->next = nullptr;
			}

		list_comp(data->left_arm->next_c->next_ra, temp, path +
"Коромысло -> Левое плечо -> ", result);
		list_comp(data->right_arm->next_c->next_ra, temp, path +
"Коромысло -> Правое плечо -> ", result);
	}
}

// - - - - - input recursion - - - - - //

std::istream& operator>>(std::istream& in, rocker_arm& data) {

	char symbol;
	in >> symbol;
	if (symbol != '(') return in;

	data.left_arm = new arm;
	in >> *data.left_arm;
```

```
	if (!data.left_arm->next_c) {

		rocker_arm_del(data);
		return in;
	}

	data.right_arm = new arm;
	in >> *data.right_arm;

	if (!data.right_arm->next_c) {

		rocker_arm_del(data);
		return in;
	}

	in >> symbol;
	if (symbol != ')') rocker_arm_del(data);
	return in;
}

std::istream& operator>>(std::istream& in, arm& data) {

	char symbol;
	in >> symbol;
	if (symbol != '(') return in;

	in >> data.lenght;
	data.next_c = new cargo;
	in >> *data.next_c;

	if (!data.next_c) {

		arm_del(data);
		return in;
	}

	in >> symbol;
	if (symbol != ')') arm_del(data);
	return in;
}

std::istream& operator>>(std::istream& in, cargo& data) {

	char symbol;
	in >> symbol;
	if (symbol != '(') {

		in.seekg(-1, in.cur);
		in >> data.weight;
```

```
	}
	else {

		in.seekg(-1, in.cur);
		data.next_ra = new rocker_arm;
		in >> *data.next_ra;

		if (!data.next_ra) cargo_del(data);
	}
	return in;
}

// - - - - - clear recursion - - - - - //

void rocker_arm_del(rocker_arm& data) {

	if (data.left_arm) {

		arm_del(*data.left_arm);
		delete data.left_arm;
		data.left_arm = nullptr;
	}
	if (data.right_arm) {

		arm_del(*data.right_arm);
		delete data.right_arm;
		data.right_arm = nullptr;
	}
}

void arm_del(arm& data) {

	if (data.next_c) {

		cargo_del(*data.next_c);
		delete data.next_c;
		data.next_c = nullptr;
	}
}

void cargo_del(cargo& data) {

	if (data.next_ra) {

		rocker_arm_del(*data.next_ra);
		delete data.next_ra;
		data.next_ra = nullptr;
	}
}
```

```
// - - - - - other functions - - - - - //

void list_print(std::ostream& out, const list& data, const std::string& str)
{

	node* head;
	int i = 1;
	for (head = data; head; head = head->next) {

		if (head->path == "")
			out << i++ << ".\t" << "Два бк равны -- " << str << "\n\n";
		if (head->path != "")
			out << i++ << ".\t" << "Корневое " << head->path << "|| ->
" << str << "\n\n";
	}
}

std::istream& operator>>(std::istream& in, s_string& data) {

	int count = 0;
	char symb;
	do {
		symb = in.get();
		if (symb == '(') count++;
		if (symb == ')') count--;
		if (symb != '\n' && symb != '\t') data.str += symb;
	} while (count || symb != ')');
	return in;
}

int kb_react(const char* first, const char* second) {

	int flag = 1;
	while (1) {

		if (flag == 1) std::cout << first;
		else std::cout << second;

		unsigned char ch = _getch();
		if (ch == 224) ch = _getch();
		switch (ch) {
		case 75:
			if (flag == 1) flag++;
			else flag--;
			break;
		case 77:
			if (flag == 1) flag++;
			else flag--;
```

```
                  break;
            case 13:
                  return flag;
            case 27:
                  return 0;
            }

            std::cout << '\r';
      }
}

void list_del(list& data) {

      for (; data;) {

            node* next = data->next;
            delete data;
            data = next;
      }
}

void menu_print(bool hat, const std::string str = "", std::ofstream* ptr =
nullptr) {

      if (hat) {

            system("cls");
            std::cout << "Программа по поиску всех вхождений заданного
бинарного коромысла в другое.\nДюков В.А. гр. 9303. Для выхода нажмите -
Esc.\n\n";
      }
      std::cout << str;
      if (ptr && ptr->is_open()) *ptr << str;
}

// - - - - - - - - - - - - - - - - - - - //

int main() {

      SetConsoleCP(1251);
      SetConsoleOutputCP(1251);

      std::string str_1, str_2;
      rocker_arm big_ra, small_ra;
      list result;
      int flag = F_START;
      std::stringstream consol;
      std::ifstream f_file;
      std::ofstream o_file("result.txt");
```

```
	while (flag != F_EXIT) {

		if (flag == F_START) {

			menu_print(true, "Выберите, откуда брать исходные
данные:\n");
			flag = kb_react(">Консоль<\t Файл ", " Консоль \t>Файл<");
			if (flag == 1) flag = F_CONSOL;
			else
				if (flag == 2) flag = F_FILE;
			continue;
		}

		if (flag == F_CONSOL) {

			menu_print(true, "Введите БинКор, в котором будет поиск
совпадений:\n\n");
			std::getline(std::cin, str_1);
			consol.str(str_1);
			consol >> big_ra;

			menu_print(false, "\n\nВведите БинКор, совпадения с которым
будут найдены:\n\n");
			std::getline(std::cin, str_2);
			consol.str(str_2);
			consol >> small_ra;

			flag = F_CRES;
		}

		if (flag == F_FILE) {

			if (!f_file.is_open()) {

				menu_print(true, "Введите имя файла:\n\n");
				std::getline(std::cin, str_1);

				f_file.open(str_1);

				if (f_file.fail()) {

					menu_print(false, "Ошибка ввода имени
файла.\nХотите повторить ввод?\n");
					flag = kb_react(">Да<\t Нет ", " Да \t>Нет<");
					if (flag == 1) flag = F_FILE;
					if (flag == 2) flag = F_START;
					continue;
				}
```

```
                }

                int now = f_file.tellg();
                f_file >> big_ra >> small_ra;
                if (big_ra.left_arm && small_ra.left_arm) {

                        s_string sstr_1, sstr_2;
                        f_file.seekg(now, f_file.beg);
                        f_file >> sstr_1 >> sstr_2;
                        str_1 = sstr_1.str;
                        str_2 = sstr_2.str;
                }

                flag = F_FRES;
        }

        if (flag == F_FRES || flag == F_CRES) {

                menu_print(true, "Исходные данные:\n\n", &o_file);
                std::cout << str_1 << " <- " << str_2 << "\n\n";
                o_file << str_1 << " <- " << str_2 << "\n\n";

                menu_print(false, "Результат выполнения программы:\n\n",
&o_file);

                if (!big_ra.left_arm || !small_ra.left_arm) {

                        if (!big_ra.left_arm)
                                menu_print(false, "Ошибка ввода БинКор, в
котором идёт поиск совпадений.\n", &o_file);
                        if (!small_ra.left_arm)
                                menu_print(false, "Ошибка ввода БинКор,
совпадения с которым в поиске.\n", &o_file);
                }
                else {

                        result = nullptr;
                        list_comp(&big_ra, small_ra, "", result);
                        if (result) {

                                list_print(std::cout, result, str_2);
                                list_print(o_file, result, str_2);
                        }
                        else menu_print(false, "Нет совпадений.\n", &o_file);
                }

                std::cout << std::endl;
                o_file << "\n_____\n"
<<std::endl;
```

```
                  rocker_arm_del(big_ra);
                  rocker_arm_del(small_ra);
                  list_del(result);

                  if (flag == F_FRES) {

                        std::cout << "Продолжить вывод из файла?\n";
                        int x = kb_react(">Да<\t Нет ", " Да \t>Нет<");
                        if (x == 1) flag = F_FILE;
                        if (x == 2) {

                              f_file.close();
                              flag = F_START;
                        }
                        continue;
                  }
                  if (flag == F_CRES) {

                        if (!big_ra.left_arm) std::cout << "Хотите повторить
ввод?\n";
                        int x = kb_react(">Да<\t Нет ", " Да \t>Нет<");
                        if (x == 1) flag = F_CONSOL;
                        if (x == 2) flag = F_START;
                        continue;
                  }
            }
      }

      o_file.close();
      return 0;
}
```