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 _____ Лойконен М.Р.

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. Получить навыки решения задач обработки иерархических списков, как с использованием базовых функций их рекурсивной обработки, так и без использования рекурсии.

Задание.

Вариант 11.

Сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращённой скобочной записью исходного иерархического списка после устранения всех внутренних скобок.

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_lis t> ::= 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) >

Выполнение работы.

Для хранения списка в программе были реализованы следующие структуры:

`two_ptr` — содержит 2 поля `s_expr* head` и `s_expr* tail`, которые являются соответственно указателями на «голову» и «хвост» списка.

`s_expr` — содержит поле `bool tag`, которое обозначает, является ли элемент списка атомом (`true`) или списком (`false`), и `union node`, которое содержит 2 поля: `base atom` — хранит атом, `two_ptr pair` — элемент структуры `two_pair`.

Для работы со списком были реализованы следующие функции:

`lisp head(const lisp s)` — возвращает «голову» списка.

`lisp tail(const lisp s)` — возвращает «хвост» списка.

`bool isAtom(const lisp s)` — возвращает `true`, если элемент списка атом, в противном случае false.

`bool isNull(const lisp s)` — возвращает `true`, если передан пустой список.

`lisp cons(const lisp h, const lisp t)` — создает новый список из двух других.

`lisp make_atom(const base x)` — создает элемент-атом.

`void destroy(const lisp s)` — удаляет список.

`void write_lisp(const lisp s)` и `void write_seq(const lisp s)` — функции для вывода сокращенной скобочной записи списка.

`void read_lisp(lisp& s, ifstream& fin, int& flag)`, `void read_s_expr(base prev, lisp& s, ifstream& fin)` и `void read_seq(lisp& s, ifstream& fin)` — функции для чтения списка из файла.

Также была реализована структура хранения линейного списка:

`list` — содержит 2 поля: `base data` — значение элемента и `list* next` — указатель на следующий элемент линейного списка.

Были реализованы следующие функции работы с линейным списком:

`list* createListEl(base el)` — создает элемент линейного списка.

`void push(list* &head, base el)` — добавляет элемент в линейный список.

`int countEl(list* head)` — считает количество элементов в линейном списке.

`void removeEl(list* head)` — удаляет последний элемент списка.

`void printList(list* head)` — печатает скобочную запись линейного списка.

Для решения задания была написана функция `void lispElements(const lisp s, list* &head)`, которая обходит все элементы иерархического списка `s`, и добавляет их в линейный список `head`.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе выполнения работы было изучено понятие иерархического списка. Были приобретены навыки по их обработке с помощью рекурсивных функций.

Была написана программа, выполняющая чтение из файла сокращенной скобочной записи иерархического списка и преобразующая его в линейный список.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

using namespace std;

typedef char base;
struct s_expr;

struct two_ptr{
	s_expr *head;
	s_expr *tail;
};

struct s_expr{
	bool tag;
	union{
		base atom;
		two_ptr pair;
	} node;
};

typedef s_expr *lisp;

lisp head(const lisp s);
lisp tail(const lisp s);
bool isAtom(const lisp s);
bool isNull(const lisp s);
lisp cons(const lisp h, const lisp t);
lisp make_atom(const base x);
void destroy(lisp s);
void write_lisp(const lisp s);
void write_seq(const lisp s);
void read_lisp(lisp& s, ifstream& fin, int& flag);
void read_s_expr(base prev, lisp& s, ifstream& fin);
void read_seq(lisp& s, ifstream& fin);

lisp head(const lisp s){
	if (s != nullptr){
		if (!isAtom(s))
			return s->node.pair.head;
		else{
			cout << "Error: s is atom\n";
			destroy(s);
			exit(1);
		}
	}
	else{
		cout << "Error: s is nullptr\n";
```

```
            exit(1);
        }
}

lisp tail(const lisp s){
        if (s != nullptr){
            if (!isAtom(s))
                return s->node.pair.tail;
            else{
                cout << "Error: s is atom\n";
                destroy(s);
                exit(1);
            }
        }
        else{
            cout << "Error: s is nullptr\n";
            exit(1);
        }
}

bool isAtom(const lisp s){
        if (s == nullptr)
            return false;
        else
            return s->tag;
}

bool isNull(const lisp s){
        return s == nullptr;
}

lisp cons(const lisp h, const lisp t){
        lisp p;
        if (isAtom(t)){
            cout << "Error\n";
            destroy(h);
            destroy(t);
            exit(1);
        }
        else{
            p = new s_expr;
            if (p == NULL){
                cout << "Error. Memory is not enough\n";
                exit(1);
            }
            else{
                p->tag = false;
                p->node.pair.head = h;
                p->node.pair.tail = t;
                return p;
            }
        }
}

lisp make_atom(const base x){
        lisp s;
```

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

void destroy(lisp s){
      if (s != nullptr){
            if (!isAtom(s)){
                  destroy(head(s));
                  destroy(tail(s));
            }
            delete s;
      }
}

void write_lisp(const lisp s){
      if (isNull(s))
            cout << "()";
      else if (isAtom(s))
            cout << ' ' << s->node.atom << ' ';
      else{
            cout << "(";
            write_seq(s);
            cout << ")";
      }
}

void write_seq(const lisp s){
      if (!isNull(s)){
            write_lisp(head(s));
            write_seq(tail(s));
      }
}

void read_lisp(lisp& s, ifstream& fin, int& flag){
      base x;
      do
            fin >> x;
      while (x == ' ');
      if (fin.eof()){
            flag = 1;
            return;
      }
      read_s_expr(x, s, fin);
}

void read_s_expr(base prev, lisp& s, ifstream& fin){
      if (prev == ')'){
            cout << "! List.Error 1\n";
            destroy(s);
            exit(1);
      }
      else if (prev != '(')
            s = make_atom(prev);
      else
```

```
            read_seq(s, fin);
}

void read_seq(lisp& s, ifstream& fin){
      base x;
      lisp p1, p2;
      if (!(fin >> x)){
            cout << "! List.Error 2\n";
            destroy(s);
            exit(1);
      }
      else{
            while (x == ' ')
                  fin >> x;
            if (x == ')')
                  s = nullptr;
            else{
                  read_s_expr(x, p1, fin);
                  read_seq(p2, fin);
                  s = cons(p1, p2);
            }
      }
}

struct list{
      base data;
      list* next;
};

list* createListEl(base el){
      list* t = new list;
      t->data = el;
      t->next = nullptr;
      return t;
}

void push(list* &head, base el){
      if (!head){
            head = createListEl(el);
      return;
      }
      list* cur = head;
      list* t = createListEl(el);
      while (cur->next)
            cur = cur->next;
      cur->next = t;
}

int countEl(list* head){
      if (!head)
            return 0;
      list* cur = head;
      int n = 0;
      while (cur){
            n++;
            cur = cur->next;
```

```
	}
	return n;
}

void removeEl(list* head){
	list* cur = head;
	list* prev = head;
	while (cur->next){
		prev = cur;
		cur = cur->next;
	}
	prev->next = nullptr;
	delete cur;
}

void printList(list* head){
	cout <<"(";
	list* cur = head;
	while(cur){
		cout << " " << cur->data << " ";
		cur = cur->next;
	}
	cout << ")\n";
}

void lispElements(const lisp s, list* &hd){
	if (isNull(s))
		return;
	if (isAtom(s)){
		push(hd, s->node.atom);
	}
	else{
		lispElements(head(s), hd);
		lispElements(tail(s), hd);
	}
}

int main(){
	string filename;
	cout << "Enter a file name\n";
	cin >> filename;
	ifstream fin(filename);
	int flag = 0;
	while(fin){
		lisp s;
		list* head = nullptr;
		read_lisp(s, fin, flag);
		if (flag)
			break;
		cout << "Иерархический список: ";
		write_lisp(s);
		cout << '\n';
		lispElements(s, head);
		cout << "Линейный список: ";
		printList(head);
		cout << '\n';
```

```
            destroy(s);
            for (int i = 0; i<countEl(head)-1; i++){
                  removeEl(head);
            }
            delete head;
      }
      fin.close();
      return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(a (b c d) e)	Иерархический список: (a (b c d) e) Линейный список: (a b c d e)	Тест пройден.
2.	(a b c d)	Иерархический список: (a b c d) Линейный список: (a b c d)	Тест пройден.
3.	(e b (c h))	Иерархический список: (e b (c h)) Линейный список: (e b c h)	Тест пройден.
4.	((g n) k d)	Иерархический список: ((g n) k d) Линейный список: (g n k d)	Тест пройден.
5.	(z (b (x (d e f)) j))	Иерархический список: (z (b (x (d e f)) j)) Линейный список: (z b x d e f j)	Тест пройден.
6.	(((a) b) c)	Иерархический список: (((a) b) c) Линейный список: (a b c)	Тест пройден.
7.	(a)	Иерархический список: (a) Линейный список: (a)	Тест пройден.
8.	()	Иерархический список: () Линейный список: ()	Тест пройден.
9.	(a b c (d e (g p (h)) e) q)	Иерархический список: (a b c (d e (g p (h)) e) q) Линейный список: (a b c d e g	Тест пройден.

		p h e q)	