**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. 6383 |  | Михайлов Ю.А. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

1. **Цель работы.**

Ознакомиться 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.

1. **Постановка задачи.**

Задание предполагает самостоятельную разработку студентом одного или нескольких модулей на языке С++, реализующих согласованный с преподавателем набор операций над иерархическими списками, а также главной программы, непосредственно решающей поставленную задачу. Предполагается выполнение задания в двух вариантах: с использованием базовых функций рекурсивной обработки иерархических списков и без использования рекурсии.

Для представления иерархических списков рекомендуется использовать сокращенную скобочную запись.

Вариант 10.3.

Пусть выражение (логическое, арифметическое) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме, т.е. **( <операция> <аргументы>)**. Аргументов может быть 1, 2 и более. Например, (+ a (\* b (- c))) или (OR a (AND b (NOT c))).

Вычислить арифметическое выражение. На вход подается список значений переменных ( (x1 c1) (x2 c2) … (xk ck) ), где xi – переменная, а ci – ее значение (константа) целого типа.

1. **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

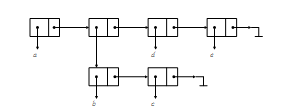
< Non\_null\_list(El) > ::= < Pair(El) >

< Pair(El) > ::= ( < Head\_l(El) > . < Tail\_l(El) > )

< Head\_l(El) > ::= < El >

< Tail\_l(El) > ::= < L\_list(El) >

Традиционно иерархические списки представляют или графически или в виде скобочной записи. На рис. 1 приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись － это (a (b c) d e).



*Рисунок 1. Пример представления иерархического списка в виде двумерного рисунка*

Переход от полной скобочной записи, соответствующей определению иерархического списка, к сокращенной производится путем отбрасывания конструкции . Nill и удаления необходимое число раз пары скобок вместе с предшествующей открывающей скобке точкой. Согласно приведенному определению иерархического списка, структура непустого иерархического списка － это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

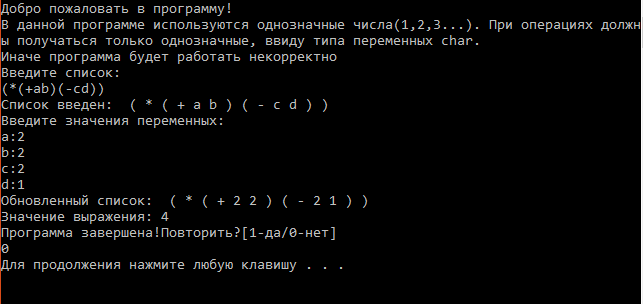
1. **Спецификация программы.**

Программа предназначена для вычисления арифметического выражения. Программа написана на языке C++. Входными данными для программы являются:

-иерархический список в сокращенной скобочной записи;

-значение переменных, которые присутствуют в списке.

1. **Пример диалога с пользователем.**

****

1. **Реализация.**

Функции:

void enter\_lisp(lisp x) – функция, в которой пользователь вводит значения для переменных списка.

lisp flatten(const lisp s) – функция выравнивания списка.

void res\_lisp(const lisp x, string& dec) – функция, которая закидывает весь список в переменную типа string для упрощения вычисления выражения.

void rezult(string& dec) – функция, которая вычисляет значение выражения, которое задано в префиксной форме.

1. **Тестирование.**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| (+ab)  1  2 | 3 |
| (\*(-ab)(+cd))  4  3  2  1 | 3 |
| (/ab)  9  3 | 3 |
| (\*(+(-ab)(+cd))(-oy))  2  1  1  1  8  6 | 6 |
| (/(+(-(+ab)(+zx))p)(\*yu))  2102742 | 1 |

**Выводы****.**

В результате выполнения лабораторной работы мною были получены основные сведения о иерархических списках. Ознакомился с базовыми функциями обработки иерархических списков. Были написаны функции, позволяющие присвоить переменным определенное значение и вычислить выражение , заданное в префиксной форме.

**Приложение А. Исходный код.**

**Extra.h**

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

struct s\_expr;

struct two\_ptr

{

s\_expr \*hd;

s\_expr \*tl;

}; //end two\_ptr;

struct s\_expr {

bool tag; // true: atom, false: pair

union

{

base atom;

two\_ptr pair;

} node; //end union node

}; //end s\_expr

typedef s\_expr \*lisp;

// функции

void print\_s\_expr(lisp s);

// базовые функции:

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);

base getAtom(const lisp s);

void read\_lisp(lisp& y);

void read\_s\_expr(base prev, lisp& y);

void read\_seq(lisp& y);

void write\_lisp(const lisp x);

void write\_seq(const lisp x);

lisp copy\_lisp(const lisp x);

#include <iostream>

#include <cstdlib>

using namespace std;

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"; system("pause"); exit(1); }

else {

cerr << "Error: Head(nil) \n";

exit(1);

}

}

bool isAtom(const lisp s)

{

if (s == NULL) return false;

else return (s->tag);

}

bool isNull(const lisp s)

{

return s == NULL;

}

lisp tail(const lisp s)

{// PreCondition: not null (s)

if (s != NULL) if (!isAtom(s)) return s->node.pair.tl;

else { cerr << "Error: Tail(atom) \n"; system("pause"); exit(1); }

else {

cerr << "Error: Tail(nil) \n";

exit(1);

}

}

lisp cons(const lisp h, const lisp t)

// PreCondition: not isAtom (t)

{

lisp p;

if (isAtom(t)) { cerr << "Error: Cons(\*, atom)\n"; system("pause"); exit(1); }

else {

p = new s\_expr;

if (p == NULL) { cerr << "Memory not enough\n"; system("pause"); exit(1); }

else {

p->tag = false;

p->node.pair.hd = h;

p->node.pair.tl = 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 != NULL) {

if (!isAtom(s)) {

destroy(head(s));

destroy(tail(s));

}

delete s;

// s = NULL;

};

}

base getAtom(const lisp s)

{

if (!isAtom(s)) { cerr << "Error: getAtom(s) for !isAtom(s) \n"; system("pause"); exit(1); }

else return (s->node.atom);

}

void read\_lisp(lisp& y)

{

base x;

do cin >> x; while (x == ' ');

read\_s\_expr(x, y);

} //end read\_lisp

void read\_s\_expr(base prev, lisp& y)

{ //prev ? ранее прочитанный символ}

if (prev == ')') { cerr << " ! List.Error 1 " << endl; system("pause"); exit(1); }

else if (prev != '(') y = make\_atom(prev);

else read\_seq(y);

} //end read\_s\_expr

void read\_seq(lisp& y)

{

base x;

lisp p1, p2;

if (!(cin >> x)) { cerr << " ! List.Error 2 " << endl; system("pause"); exit(1); }

else {

while (x == ' ') cin >> x;

if (x == ')') y = NULL;

else {

read\_s\_expr(x, p1);

read\_seq(p2);

y = cons(p1, p2);

}

}

} //end read\_seq

void write\_lisp(const lisp x)

{//пустой список выводится как ()

if (isNull(x)) cout << " ()";

else if (isAtom(x)) cout << ' ' << x->node.atom;

else { //непустой список}

cout << " (";

write\_seq(x);

cout << " )";

}

} // end write\_lisp

void write\_seq(const lisp x)

{//выводит последовательность элементов списка без обрамляющих его скобок

if (!isNull(x)) {

write\_lisp(head(x));

write\_seq(tail(x));

}

}

lisp copy\_lisp(const lisp x)

{

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)));

} //end copy-lisp

**main.cpp**

#include <iostream>

#include <cstdlib>

#include "Extrar.h"

#include <windows.h>

#include <string>

using namespace std;

lisp concat(const lisp y, const lisp z);

void enter\_lisp(const lisp x);

void enter\_seq(const lisp x);

void res\_lisp(const lisp x,string& dec);

void res\_seq(const lisp x,string& dec);

lisp flatten(const lisp s);

void rezult(string& dec);

bool xnak = false;

void start\_menu() {

string dec;

lisp s1;

int c = 0;

cout << boolalpha;

cout << "Введите список:" << endl;

read\_lisp(s1);

cout << "Список введен: ";

write\_lisp(s1);

cout << endl;

cout << "Введите значения переменных:" << endl;

enter\_lisp(s1);

cout << "Обновленный список: ";

write\_lisp(s1);

res\_lisp(s1, dec);

cout << endl;

cout << "Значение выражения: ";

rezult(dec);

cout << dec << endl;

destroy(s1);

cout << "Программа завершена!Повторить?[1-да/0-нет]" << endl;

}

int main()

{

int c = 0;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Добро пожаловать в программу!" << endl;

cout << "В данной программе используются однозначные числа(1,2,3...). При операциях должны получаться только однозначные, ввиду типа переменных char." << endl;

cout << "Иначе программа будет работать некорректно" << endl;

start\_menu();

cin >> c;

while (c == 1) {

system("cls");

start\_menu();

cin >> c;

}

system("pause");

return 0;

}

void rezult(string& dec) {//вычисление выражения в префиксной форме рекурсивным методом

for (int i = 0; i < dec.length() - 2; i++) {

if (dec[i] == '+' || dec[i] == '\*' || dec[i] == '-' || dec[i] == '/') {

xnak = true;

if (dec[i + 1] != '+' && dec[i+1] != '\*' && dec[i+1] != '-' && dec[i+1] != '/') {

char c = dec[i];

dec[i] = dec[i + 1];

dec[i + 1] = c;

int a, b;

a = dec[i] - '0';

b = dec[i + 2] - '0';

switch (dec[i + 1])

{

case '+':a = a + b;

dec[i] = a + '0';

break;

case '-':a = a - b;

dec[i] = a + '0';

break;

case '\*':a = a \* b;

dec[i] = a + '0';

break;

case '/':a = a / b;

dec[i] = a + '0';

break;

default:

break;

}

dec[i + 1] = dec[i + 2];

for (int j = i + 1; j < dec.length() - 1; j++) {

dec[j] = dec[j + 1];

}

for (int j = i + 1; j < dec.length() - 1; j++) {

dec[j] = dec[j + 1];

}

dec.resize(dec.length() - 2);

}

}

else break;

}

if (!xnak) {

cout << "Ошибка!Некорректно введено выражение!Нет операции!(+ - \* /)" << endl; system("pause"); exit(0);

}

if (dec.length() > 1) rezult(dec);

}

lisp flatten(const lisp s)

{//выравнивание списка

if (isNull(s)) return NULL;

else if (isAtom(s)) return cons(make\_atom(getAtom(s)), NULL);

else //s ? непустой список

if (isAtom(head(s))) return cons(make\_atom(getAtom(head(s))), flatten(tail(s)));

else //Not Atom(Head(s))

return concat(flatten(head(s)), flatten(tail(s)));

} // end flat

void res\_lisp(const lisp x, string& dec)

{//для удобства вычисления выражения в префиксной форме, заносим его в переменную типа string

char a;

if (isNull(x)) cout << " ()";

else if (isAtom(x)) {

dec = dec + x->node.atom;

}

else { //непустой список}

res\_seq(x, dec);

}

} // end write\_lisp

void res\_seq(const lisp x,string& dec)

{//

if (!isNull(x)) {

res\_lisp(head(x),dec);

res\_seq(tail(x),dec);

}

}

void enter\_lisp(lisp x)

{//заполнение списка собственными значениями

int s;

char y;

if (isNull(x)) cout << "Пустой список!";

else if (isAtom(x)) {

if (x->node.atom != '+' && x->node.atom != '-' && x->node.atom != '\*' && x->node.atom != '/') {

cout << x->node.atom << ":";

cin >> y;

x->node.atom = y;

}

}

else {

enter\_seq(x);

}

} // end write\_lisp

void enter\_seq(lisp x)

{//

if (!isNull(x)) {

enter\_lisp(head(x));

enter\_seq(tail(x));

}

}

lisp concat(const lisp y, const lisp z)

{

if (isNull(y)) return copy\_lisp(z);

else return cons(copy\_lisp(head(y)), concat(tail(y), z));

} // end concat