**Отчёт по лабораторной работе** №\_\_\_\_\_\_\_ по курсу\_\_\_\_\_\_\_

студент группы М8О-107Б-18 Гамов Павел Антонович № по списку4

Адрес (e-mail) pagamov@gmail.com

Работа выполнена: “ “ \_\_\_\_\_\_\_ 2018г.

Преподаватель: асп. Каф. 806 Ридли А.Н.

Входной контроль знаний с оценкой

Отчёт сдан “ “ \_\_\_\_\_\_ 2018 г итоговая оценка

Подпись преподавателя

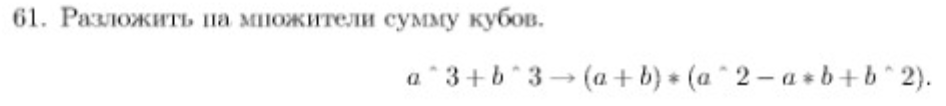
1. Тема:

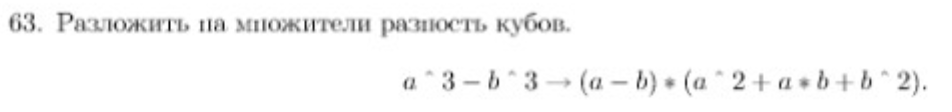
Язык Си. Алгоритмы и структуры данных. Динамическое программирование. Деревья их реализация.

1. Цель работы:

Составить программу выполнения преобразований арифметических выражений с применением деревьев. Преобразование выражения в дерево рекомендуется осуществлять одним их известных методов (Рутисхаузера или Дейкстры). Операнды целочисленного типа. Реализовать в виде набора подпрограмм.

1. Задание: 61 и 63





1. **Оборудование** (лабораторное):

ЭВМ Pentium , процессор2.7 ГГц , имя узла сети с ОП 4096 МБ

НМД 2 ГБ. Терминал bash адрес . Принтер

Другие устройства

Оборудование ПЭВМ студента, если использовалось:

Процессор 2.7 GHz Intel Core i5, ОП 8 GB (1867 MHz LPDDR3), НМД 256 ГБ. Монитор встроенный

Другие устройства

1. Программное обеспечение (лабораторное):

Операционная система семейства Unix , наименование Ubuntu версия 18.04.1 LTS

Интерпретатор команд bash версия 4.3.48

Система программирования версия

Редактор текстов Emacs версия 7.4

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных

1. Программное обеспечение ЭВМ студента, если использовалось:

Операционная система семейства Unix , наименование Ubuntu версия 18.04.1 LTS

Интерпретатор команд bash версия 4.3.48

Система программирования версия

Редактор текстов GNU Emacs версия 25.2.2

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных /home/pavel

1. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальное описание с пред- и постусловиями)

Алгоритм сортировочной станции — способ разбора математических выражений, представленных в инфиксной нотации. Может быть использован для получения вывода в виде обратной польской нотации или в виде абстрактного синтаксического дерева. Алгоритм изобретен Эдсгером Дейкстрой и назван им «алгоритм сортировочной станции», поскольку напоминает действие железнодорожной сортировочной станции.

Так же, как и вычисление значений выражений в обратной польской записи, алгоритм работает при помощи стека. Инфиксная запись математических выражений чаще всего используется людьми, её примеры: 2+4 и 3+6\*(3-2). Для преобразования в обратную польскую нотацию используется 2 строки: входная и выходная, и стек для хранения операторов, еще не добавленных в выходную очередь. При преобразовании алгоритм считывает 1 символ и производит действия, зависящие от данного символа.

Оригинальное описание алгоритма - http://www.cs.utexas.edu/~EWD/MCReps/MR35.PDF

Пункты 1-7 отчёта составляются **строго** **до** начала лабораторной работы.

Допущен к выполнению работы. Подпись преподавателя

1. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты, либо соображения по тестированию].

Разобьем данную задачу на подзадачи.

Первым делом необходимо реализовать структуру данных типа стек для деревьев внутри, которые в свою очередь являются бинарными, содержащие в себе строки.

Далее алгоритм автомата конечных состояний для записи посимвольно в стеки. Данных алгоритм дает в результате стек, состоящий из деревьев глубины 1. При ошибке программа сразу выдает несоответствие записи и чистить программу. Далее второй этап и это функция, принимающая на вход стек и дающая обратно дерево. Функция не работает на неправильном стеке, за безопасность отвечает предыдущий этап.

Пробегая по нодам, программа соединяет, что возможно, увеличивая глубину дерева и создавая в конечном итоге стек с одним деревом в нем, удаляет стек, возвращает дерево.

Этап третий. Функция рекурсивного обхода и нахождения шаблона в дереве. Проверяем условие шаблона для нода, если верно, копируем поддеревья и разбиваем все на части. Для создания нового дерева пишется скрипт шаблон, который на основании уже используемых функций, пушит/добавляет в стек в нужном порядке нужные деревья или элементы, далее функция из второго этапа соединяет данный стек в единое дерево, которым мы заменяем предыдущее поддереве найденного нами нода.

Так как поддеревья могут являться также нужными к замене, рекурсивно обходим все до терминальных нодов.

Заметим, что алгоритм пункта три не работает на неявных шаблонах данных вариантов. К примеру, если степень 3 задана как 5-2, алгоритм пропустит данный нод.

Четвертый пункт программы. написания функции которая считает дерево если возможно. Рекурсивная функция спускается до терминальных нодов и в зависимости от шаблона, используя функции перевода char массивов в int и обратно, собирает и удаляет поддеревья, позволяя посчитать результат операций.

Конец программы. полная чистка памяти и освобождение всего что можно.

Как вариант дополнить программу, написать интерфейс дозаписи переменных в программу, для того чтобы можно было решать уравнения относительно введенных неизвестных.

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с текстовыми примерами, подписанный преподавателем)
2. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб.  или  дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
| 1 | дом | .2018 |  |  |  |  |

1. Замечание автора по существу работы
2. Выводы

Работа с различными структурами данных позволяет развить свои навыки в написании программ на языке Си. Данная работа является очень интересной как с точки зрения теории, так и с точки зрения реализации.

1. **Недочеты**, допущенные при выполнении задания, могут быть устранены следующим образом

Подпись студента

**MacBook-Pro-Pavel:24laba pavelgamov$ cat makefile**

CC = gcc

CCFLAGS = -Wall -pedantic -std=c99

###\_\_\_\_###

program: main.o list.o tree.o array.o ; $(CC) $(CCFLAGS) -g main.o list.o tree.o array.o -o program

main.o: main.c list.h tree.h array.h ; $(CC) $(CCFLAGS) -c -g main.c

list.o: list.c list.h tree.h array.h ; $(CC) $(CCFLAGS) -c -g list.c

array.o: array.c array.h ; $(CC) $(CCFLAGS) -c -g array.c

tree.o: tree.c tree.h array.h list.h ; $(CC) $(CCFLAGS) -c -g tree.c

clean: ; rm \*.o

###\_\_\_###

**MacBook-Pro-Pavel:24laba pavelgamov$ cat array.h**

#ifndef ARRAY\_H

#define ARRAY\_H

#include <stdio.h>

#include <stdlib.h>

//structure

enum Status {

operand,

operator,

vait,

left,

right

};

//functions

char \* addToDataArray(char \* data, char new);

char \* createArray(char \* arr);

char \* refresh(char \* arr);

int is\_operatorNorm(char q);

int is\_operandLetter(char q);

int is\_opereandNumber(char q);

int is\_operatorScob(char q);

int ratingOperator(char in, char sver);

int is\_needToDeAdded(char \* arr);

int is\_needToDeAddedMinus(char \* arr);

int is\_integer(char \* arr);

int replaceCharToInt(char \* arr);

char \* replaceIntToChar(int number);

int int\_power(int low, int pow);

#endif

**MacBook-Pro-Pavel:24laba pavelgamov$ cat array.c**

#include "array.h"

char \* addToDataArray(char \* data, char new)

{

int size = 0;

while (data[size] != '\0')

size++;

data = (char \*)realloc(data, (size + 2) \* sizeof(char));

data[size] = new;

data[size + 1] = '\0';

return data;

}

char \* createArray(char \* arr)

{

arr = (char \*)malloc(sizeof(char));

arr[0] = '\0';

return arr;

}

char \* refresh(char \* arr)

{

arr = (char \*)realloc(arr, sizeof(char));

arr[0] = '\0';

return arr;

}

int is\_operatorNorm(char q)

{

if (q == '+' || q == '/' || q == '-' || q == '\*' || q == '^')

return 1;

return 0;

}

int is\_operandLetter(char q)

{

if (q >= 'a' && q <= 'z')

return 1;

return 0;

}

int is\_opereandNumber(char q)

{

if (q >= '0' && q <= '9')

return 1;

return 0;

}

int is\_operatorScob(char q)

{

if (q == '(')

return 1;

else if (q == ')')

return 2;

return 0;

}

int ratingOperator(char in, char sver)

{

switch (in) {

case '^':

return 1;

break;

case '/':

if (sver == '-' || sver == '+' || sver == '(')

return 1;

break;

case '\*':

if (sver == '-' || sver == '+' || sver == '(')

return 1;

break;

case '+':

if (sver == '(')

return 1;

break;

case '-':

if (sver == '(')

return 1;

break;

}

return 0;

}

int int\_power(int low, int pow)

{

int a = low;

if (pow == 0)

return 1;

while (pow - 1 != 0) {

a \*= low;

pow--;

}

return a;

}

int is\_needToDeAdded(char \* arr)

{

if (is\_operandLetter(arr[0]) || is\_opereandNumber(arr[0]) || is\_operandLetter(arr[1]) || is\_opereandNumber(arr[1]))

return 1;

return 0;

}

int is\_needToDeAddedMinus(char \* arr)

{

if (arr[0] == '-' && (is\_operandLetter(arr[1]) || is\_opereandNumber(arr[1])))

return 0;

if (is\_operatorNorm(arr[0]))

return 1;

return 0;

}

int is\_integer(char \* arr)

{

int i = 0;

if ((arr[0] == '-' && arr[1] == '\0') || is\_operandLetter(arr[0]) || is\_operatorNorm(arr[0]))

return 0;

i++;

while (arr[i] != '\0') {

if (!(is\_opereandNumber(arr[i])))

return 0;

i++;

}

return 1;

}

int replaceCharToInt(char \* arr) {

int res = 0, i = 0, minus = 0;

if (arr[i] == '-') {

minus = 1;

i += 1;

}

while (arr[i] != '\0') {

switch (arr[i]) {

case '0': break;

case '1': res += 1; break;

case '2': res += 2; break;

case '3': res += 3; break;

case '4': res += 4; break;

case '5': res += 5; break;

case '6': res += 6; break;

case '7': res += 7; break;

case '8': res += 8; break;

case '9': res += 9; break;

}

res \*= 10;

i++;

}

res /= 10;

if (minus)

res \*= -1;

return res;

}

char \* replaceIntToChar(int number)

{

char \* arr = NULL;

arr = createArray(arr);

int pivet = number, size = 0;

if (number == 0) {

addToDataArray(arr, '0');

return arr;

}

if (number < 0) {

number \*= -1;

addToDataArray(arr, '-');

}

while (pivet != 0) {

pivet /= 10;

size++;

}

for (int i = 0; i < size; i++) {

pivet = number;

if (i == 0)

pivet = pivet / int\_power(10, size - 1);

else

pivet = pivet / int\_power(10, size - i - 1) % 10;

switch (pivet) {

case 0: arr = addToDataArray(arr, '0'); break;

case 1: arr = addToDataArray(arr, '1'); break;

case 2: arr = addToDataArray(arr, '2'); break;

case 3: arr = addToDataArray(arr, '3'); break;

case 4: arr = addToDataArray(arr, '4'); break;

case 5: arr = addToDataArray(arr, '5'); break;

case 6: arr = addToDataArray(arr, '6'); break;

case 7: arr = addToDataArray(arr, '7'); break;

case 8: arr = addToDataArray(arr, '8'); break;

case 9: arr = addToDataArray(arr, '9'); break;

}

}

return arr;

}

**MacBook-Pro-Pavel:24laba pavelgamov$ cat list.h**

#ifndef LIST\_H

#define LIST\_H

#include <stdio.h>

#include <stdlib.h>

#include "tree.h"

//structure

typedef struct Node {

Tree \* tree;

struct Node \* next;

struct Node \* prev;

} Node;

typedef struct List {

Node \* head;

Node \* tail;

int size;

} List;

//functions

List \* createList();

char \* popList(List \* list);

void pushList(List \* list, char \* input);

char \* shiftList(List \* list);

void unshiftList(List \* list, char \* input);

void printList(List \* list);

int notEmpty(List \* list);

void deleteList(List \*\* list);

Tree \* deleteNode(List \* list, Node \* node);

Tree \* createOperandTree(List \*\* list);

void pushListTree(List \* list, Tree \* tree);

#endif

**MacBook-Pro-Pavel:24laba pavelgamov$ cat list.c**

#include "list.h"

#include "array.h"

List \* createList()

{

List \* list = (List \*)malloc(sizeof(List));

list->head = NULL;

list->tail = NULL;

list->size = 0;

return list;

}

char \* popList(List \* list)

{

if (list->size == 0) {

return NULL;

} else if (list->size == 1) {

char \* res = list->tail->tree->data;

free(list->tail->tree);

free(list->tail);

list->head = NULL;

list->tail = NULL;

list->size--;

return res;

} else {

char \* res = list->tail->tree->data;

free(list->tail->tree);

list->tail = list->tail->prev;

free(list->tail->next);

list->tail->next = NULL;

list->size--;

return res;

}

}

void pushList(List \* list, char \* input)

{

Node \* tmp = (Node \*)malloc(sizeof(Node));

tmp->tree = createNode(input);

if (list->size == 0) {

list->head = tmp;

list->tail = tmp;

tmp->next = NULL;

tmp->prev = NULL;

} else {

tmp->next = NULL;

tmp->prev = list->tail;

list->tail->next = tmp;

list->tail = tmp;

}

list->size++;

}

int notEmpty(List \* list)

{

if (list->size != 0)

return 1;

return 0;

}

void pushListTree(List \* list, Tree \* tree)

{

Node \* tmp = (Node \*)malloc(sizeof(Node));

tmp->tree = tree;

if (list->size == 0) {

list->head = tmp;

list->tail = tmp;

tmp->next = NULL;

tmp->prev = NULL;

} else {

tmp->next = NULL;

tmp->prev = list->tail;

list->tail->next = tmp;

list->tail = tmp;

}

list->size++;

}

char \* shiftList(List \* list)

{

if (list->size == 0) {

return NULL;

} else if (list->size == 1) {

char \* res = list->head->tree->data;

free(list->head->tree);

free(list->head);

list->head = NULL;

list->tail = NULL;

list->size--;

return res;

} else {

char \* res = list->head->tree->data;

free(list->head->tree);

list->head = list->head->next;

free(list->head->prev);

list->head->prev = NULL;

list->size--;

return res;

}

}

void unshiftList(List \* list, char \* input)

{

Node \* tmp = (Node \*)malloc(sizeof(Node));

tmp->tree = createNode(input);

if (list->size == 0) {

list->head = tmp;

list->tail = tmp;

tmp->next = NULL;

tmp->prev = NULL;

} else {

tmp->next = list->head;

list->head->prev = tmp;

tmp->prev = NULL;

list->head = tmp;

}

list->size++;

}

void printList(List \* list)

{

printf("list: ");

if (list->size != 0) {

Node \* tmp = list->head;

int j = 0;

for (int i = 0; i < list->size; i++) {

while (tmp->tree->data[j] != '\0') {

printf("%c", tmp->tree->data[j]);

j++;

}

j = 0;

putchar(' ');

tmp = tmp->next;

}

putchar('\n');

}

}

Tree \* deleteNode(List \* list, Node \* node)

{

Tree \* res = node->tree;

if (list->head == node) {

list->head = node->next;

node->next->prev = NULL;

} else if (list->tail == node) {

list->tail = node->prev;

node->prev->next = NULL;

} else {

node->prev->next = node->next;

node->next->prev = node->prev;

}

free(node);

list->size--;

return res;

}

void deleteList(List \*\* list)

{

if ((\*list) != NULL) {

int size = (\*list)->size;

Node \* tmp = (\*list)->head;

for (int i = 0; i < size; i++) {

deleteTree(&(tmp->tree));

if ((\*list)->size == 1) {

free((\*list)->head);

(\*list)->head = NULL;

(\*list)->tail = NULL;

(\*list)->size--;

} else {

tmp = tmp->next;

(\*list)->head = tmp;

free(tmp->prev);

tmp->prev = NULL;

(\*list)->size--;

}

}

free(\*list);

\*list = NULL;

}

}

Tree \* createOperandTree(List \*\* list)

{

Node \* tmp = (\*list)->head;

Tree \* left;

Tree \* right;

while ((\*list)->size != 1) {

if (is\_needToDeAdded(tmp->tree->data) || treeGoDeep(tmp->tree)) {

tmp = tmp->next;

if (is\_needToDeAdded(tmp->tree->data) || treeGoDeep(tmp->tree)) {

tmp = tmp->next;

if (is\_needToDeAddedMinus(tmp->tree->data) && !(treeGoDeep(tmp->tree))) {

left = deleteNode((\*list), tmp->prev->prev);

right = deleteNode((\*list), tmp->prev);

tmp->tree->left = left;

tmp->tree->right = right;

tmp = (\*list)->head;

} else {

tmp = tmp->prev;

}

}

} else {

tmp = tmp->next;

}

}

Tree \* res = (\*list)->head->tree;

free((\*list)->head);

free(\*list); (\*list) = NULL;

return res;

}

**MacBook-Pro-Pavel:24laba pavelgamov$ cat tree.h**

#ifndef TREE\_H

#define TREE\_H

#include <stdio.h>

#include <stdlib.h>

//struct

typedef struct Tree {

char \* data;

int size;

struct Tree \* left;

struct Tree \* right;

} Tree;

//functions

Tree \* createTree();

void printTree(Tree \* tree);

Tree \* createNode(char \* data);

void deleteTree(Tree \*\* tree);

int treeGoDeep(Tree \* tree);

Tree \* copyTree(Tree \* tree);

void reBuildTree(Tree \*\* tree);

void calculateAsUCan(Tree \*\* tree);

#endif

**MacBook-Pro-Pavel:24laba pavelgamov$ cat tree.c**

#include "tree.h"

#include "array.h"

#include "list.h"

Tree \* createTree() {

return NULL;

}

Tree \* createNode(char \* data)

{

Tree \* node = (Tree \*)malloc(sizeof(Tree));

int size = 0;

while (data[size] != '\0')

size++;

node->data = (char \*)malloc((size + 1) \* sizeof(char));

for (int i = 0; i < size; i++)

node->data[i] = data[i];

node->data[size] = '\0';

//node->size = size;

node->left = NULL;

node->right = NULL;

return node;

}

void print\_sup(Tree \* node, int deep)

{

for (int i = 0; i < deep - 1; i++)

printf(" ");

if (deep != 0)

printf("|>");

printf("%s\n", node->data);

if (node->right != NULL)

print\_sup(node->right, deep + 1);

if (node->left != NULL)

print\_sup(node->left, deep + 1);

}

void printTree(Tree \* tree)

{

if (tree == NULL)

printf("no tree\n");

else

print\_sup(tree, 0);

}

void del\_node\_supp(Tree \*\* tree)

{

if ((\*tree)->right != NULL)

del\_node\_supp(&((\*tree)->right));

if ((\*tree)->left != NULL)

del\_node\_supp(&((\*tree)->left));

free((\*tree)->data);

free(\*tree);

\*tree = NULL;

}

void deleteTree(Tree \*\* tree)

{

if (\*tree != NULL)

del\_node\_supp(tree);

}

int treeGoDeep(Tree \* tree)

{

if (tree->left != NULL || tree->right != NULL) {

return 1;

}

return 0;

}

Tree \* copyTree(Tree \* tree)

{

Tree \* tmp = createNode(tree->data);

if (tree->left != NULL)

tmp->left = copyTree(tree->left);

if (tree->right != NULL)

tmp->right = copyTree(tree->right);

return tmp;

}

int needDergee(Tree \* tree)

{

if (is\_operatorNorm(tree->data[0]) && (tree->data[0] == '+' || tree->data[0] == '-') && (!(is\_operandLetter(tree->data[1]))) && (!(is\_opereandNumber(tree->data[1]))))

if (tree->left->data[0] == '^' && is\_operatorNorm(tree->left->data[0]) && tree->right->data[0] == '^' && is\_operatorNorm(tree->right->data[0]))

if ((is\_needToDeAdded(tree->left->left->data) || treeGoDeep(tree->left->left)) && (is\_needToDeAdded(tree->right->left->data) || treeGoDeep(tree->right->left)))

if (tree->left->right->data[0] == '3' && tree->left->right->data[1] == '\0' && tree->right->right->data[0] == '3' && tree->right->right->data[1] == '\0') {

if (tree->data[0] == '+')

return 1;

else if (tree->data[0] == '-')

return 2;

}

return 0;

}

void reBuildTree(Tree \*\* tree) {

int res = needDergee(\*tree);

if (res) {

char \* arr = NULL;

arr = createArray(arr);

List \* list = createList();

pushListTree(list, copyTree((\*tree)->left->left)); // add a

pushListTree(list, copyTree((\*tree)->right->left)); // add b

if (res == 1)

pushList(list, (\*tree)->data); // add +

else if (res == 2)

pushList(list, (\*tree)->data); // add -

pushListTree(list, copyTree((\*tree)->left->left)); // add a

arr = addToDataArray(arr, '2');

pushList(list, arr); // add 2

arr = refresh(arr);

pushList(list, (\*tree)->left->data); // add ^

pushListTree(list, copyTree((\*tree)->left->left)); // add a

pushListTree(list, copyTree((\*tree)->right->left)); // add b

arr = addToDataArray(arr, '\*');

pushList(list, arr); // add \*

arr = refresh(arr);

if (res == 1)

arr = addToDataArray(arr, '-'); // add -

else if (res == 2)

arr = addToDataArray(arr, '+'); // add +

pushList(list, arr);

arr = refresh(arr);

pushListTree(list, copyTree((\*tree)->right->left)); // add b

arr = addToDataArray(arr, '2');

pushList(list, arr); // add 2

arr = refresh(arr);

pushList(list, (\*tree)->left->data); // add ^

arr = addToDataArray(arr, '+');

pushList(list, arr); // add -

arr = refresh(arr);

arr = addToDataArray(arr, '\*');

pushList(list, arr); // add \*

arr = refresh(arr);

deleteTree(tree);

\*tree = createOperandTree(&list);

free(arr);

}

if ((\*tree)->left != NULL)

reBuildTree(&((\*tree)->left));

if ((\*tree)->right != NULL)

reBuildTree(&((\*tree)->right));

}

void calculateAsUCan(Tree \*\* tree)

{

int left, right, res;

if (is\_operatorNorm((\*tree)->data[0]) && (\*tree)->data[1] == '\0') {

if (((\*tree)->left) != NULL)

calculateAsUCan(&((\*tree)->left));

if (((\*tree)->right) != NULL)

calculateAsUCan(&((\*tree)->right));

if (is\_integer((\*tree)->left->data) && is\_integer((\*tree)->right->data)) {

left = replaceCharToInt((\*tree)->left->data);

right = replaceCharToInt((\*tree)->right->data);

switch ((\*tree)->data[0]) {

case '+': res = left + right; break;

case '-': res = left - right; break;

case '^': res = int\_power(left, right); break;

case '/': res = left / right; break;

case '\*': res = left \* right; break;

}

deleteTree(&((\*tree)->left));

deleteTree(&((\*tree)->right));

free((\*tree)->data);

(\*tree)->data = replaceIntToChar(res);

}

}

}

**MacBook-Pro-Pavel:24laba pavelgamov$ cat main.c**

#include "list.h"

#include "tree.h"

#include "array.h"

//global vars section

char \* arr = NULL;

char \* pivet = NULL;

int size = 0, error = 0;

int main()

{

char q = '\0';

List \* OperatorArr = createList();

List \* OutputArr = createList();

enum Status status = vait;

arr = createArray(arr);

while (q != '\n') {

q = getchar();

if (is\_operatorScob(q) == 1) {

switch (status) {

case operand:

if (arr[0] == '-' && arr[1] == '\0') {

arr = refresh(arr);

arr = addToDataArray(arr, '0');

pushList(OutputArr, arr);

arr = refresh(arr);

arr = addToDataArray(arr, '-');

pushList(OperatorArr, arr);

arr = refresh(arr);

}

break;

case operator:

case vait:

case left:

arr = addToDataArray(arr, q);

pushList(OperatorArr, arr);

arr = refresh(arr);

status = left;

break;

default:

error = 1;

}

} else if (is\_operatorScob(q) == 2) {

switch (status) {

case operand:

pushList(OutputArr, arr);

arr = refresh(arr);

pivet = popList(OperatorArr);

while (pivet[0] != '(') {

pushList(OutputArr, pivet);

free(pivet);

pivet = popList(OperatorArr);

}

free(pivet);

status = right;

break;

case right:

pivet = popList(OperatorArr);

while (pivet[0] != '(') {

pushList(OutputArr, pivet);

free(pivet);

pivet = popList(OperatorArr);

}

free(pivet);

status = right;

break;

default:

error = 1;

}

} else if (is\_operatorNorm(q)) {

switch (status) {

case operand:

pushList(OutputArr, arr);

arr = refresh(arr);

arr = addToDataArray(arr, q);

if (notEmpty(OperatorArr)) {

pivet = popList(OperatorArr);

if (ratingOperator(q, pivet[0])) {

pushList(OperatorArr, pivet);

pushList(OperatorArr, arr);

} else {

pushList(OperatorArr, arr);

pushList(OutputArr, pivet);

}

free(pivet);

} else {

pushList(OperatorArr, arr);

}

arr = refresh(arr);

status = operator;

break;

case operator:

break;

case vait:

case left:

if (q == '-') {

arr = addToDataArray(arr, q);

status = operand;

} else if (q == '+')

break;

else

error = 1;

break;

case right:

arr = addToDataArray(arr, q);

if (notEmpty(OperatorArr)) {

pivet = popList(OperatorArr);

if (ratingOperator(q, pivet[0])) {

pushList(OperatorArr, pivet);

pushList(OperatorArr, arr);

} else {

pushList(OperatorArr, arr);

pushList(OutputArr, pivet);

}

free(pivet);

} else {

pushList(OperatorArr, arr);

}

arr = refresh(arr);

status = operator;

break;

default:

error = 1;

}

} else if (is\_operandLetter(q) || is\_opereandNumber(q)) {

switch (status) {

case operand:

arr = addToDataArray(arr, q);

break;

case operator:

case vait:

case left:

arr = addToDataArray(arr, q);

status = operand;

break;

default:

error = 1;

}

}

}

switch (status) {

case operand:

pushList(OutputArr, arr);

case right:

while (notEmpty(OperatorArr)) {

pivet = popList(OperatorArr);

pushList(OutputArr, pivet);

free(pivet);

}

break;

default:

error = 1;

}

free(arr);

if (error) {

printf("error\n");

deleteList(&OperatorArr);

deleteList(&OutputArr);

return 0;

}

deleteList(&OperatorArr);

//

printList(OutputArr);

//

puts("entered");

Tree \* tree = createOperandTree(&OutputArr);

printTree(tree);

//

puts("reBuilded");

reBuildTree(&tree);

printTree(tree);

//

puts("calculated");

calculateAsUCan(&tree);

printTree(tree);

//

deleteTree(&tree);

return 0;

}

MacBook-Pro-Pavel:24laba pavelgamov$ make

gcc -Wall -pedantic -std=c99 -c -g main.c

gcc -Wall -pedantic -std=c99 -c -g list.c

gcc -Wall -pedantic -std=c99 -c -g tree.c

gcc -Wall -pedantic -std=c99 -c -g array.c

gcc -Wall -pedantic -std=c99 -g main.o list.o tree.o array.o -o program

MacBook-Pro-Pavel:24laba pavelgamov$ ./program

((3/2)^3-10^3)^3+((10^2)^3-4^3)^3

list: 3 2 / 3 ^ 10 3 ^ - 3 ^ 10 2 ^ 3 ^ 4 3 ^ - 3 ^ +

entered

+

|>^

|>3

|>-

|>^

|>3

|>4

|>^

|>3

|>^

|>2

|>10

|>^

|>3

|>-

|>^

|>3

|>10

|>^

|>3

|>/

|>2

|>3

reBuilded

\*

|>+

|>^

|>2

|>\*

|>+

|>^

|>2

|>4

|>+

|>\*

|>4

|>^

|>2

|>10

|>^

|>2

|>^

|>2

|>10

|>-

|>4

|>^

|>2

|>10

|>-

|>\*

|>\*

|>+

|>^

|>2

|>4

|>+

|>\*

|>4

|>^

|>2

|>10

|>^

|>2

|>^

|>2

|>10

|>-

|>4

|>^

|>2

|>10

|>\*

|>+

|>^

|>2

|>10

|>+

|>\*

|>10

|>/

|>2

|>3

|>^

|>2

|>/

|>2

|>3

|>-

|>10

|>/

|>2

|>3

|>^

|>2

|>\*

|>+

|>^

|>2

|>10

|>+

|>\*

|>10

|>/

|>2

|>3

|>^

|>2

|>/

|>2

|>3

|>-

|>10

|>/

|>2

|>3

|>+

|>\*

|>+

|>^

|>2

|>4

|>+

|>\*

|>4

|>^

|>2

|>10

|>^

|>2

|>^

|>2

|>10

|>-

|>4

|>^

|>2

|>10

|>\*

|>+

|>^

|>2

|>10

|>+

|>\*

|>10

|>/

|>2

|>3

|>^

|>2

|>/

|>2

|>3

|>-

|>10

|>/

|>2

|>3

calculated

-863784208

MacBook-Pro-Pavel:24laba pavelgamov$ ./program

a-b+c-((10^3-3^3)/2)

list: a b - c + 10 3 ^ 3 3 ^ - 2 / -

entered

-

|>/

|>2

|>-

|>^

|>3

|>3

|>^

|>3

|>10

|>+

|>c

|>-

|>b

|>a

reBuilded

-

|>/

|>2

|>\*

|>+

|>^

|>2

|>3

|>+

|>\*

|>3

|>10

|>^

|>2

|>10

|>-

|>3

|>10

|>+

|>c

|>-

|>b

|>a

calculated

-

|>486

|>+

|>c

|>-

|>b

|>a