**Отчет по лабораторной работе № 24** по курсу “Фундаментальная информатика”

Студент группы М80-103Б-21 Катин Иван Вячеславович, № по списку 12

Контакты e-mail: ikatin.2003.sokol@gmail.com, telegram: @Dazz1e

Работа выполнена: «2» мая 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » \_\_\_\_\_\_\_\_\_20\_\_ г., итоговая оценка \_\_\_\_\_\_\_\_

Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Тема:** Дерево выражений

1. **Цель работы:** Научиться реализовать деревья, записывать выражение в бин деререво и преоброзовывать их.
2. **Задание:** 24. Перемножить дроби.
3. **Оборудование** (студента):

Процессор *Intel® Core™ i5-9300H CPU @ 2.40GHz × 8* с ОП 7,6 GiB, НМД *1024* Гб. Монитор *1920x1080*

1. **Программное обеспечение (**студента**):**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04.3 LTS*

интерпретатор команд: *bash* версия *4.4.20(1)-release*.

Система программирования -- CLion--**,** редактор текстов *emacs* версия *25.2.2*

Утилиты операционной системы --

Прикладные системы и программы – **LibreOffice**

Местонахождение и имена файлов программ и данных на домашнем компьютере – *home*/dazzle

**6**. **Идея, метод, алгоритм.** Реализовать дерево, вектор и токен. Токену содержит информацию о типе+имя оператора, значение или имя переменной.

Типы: operator, variable, bracket, floating,integer, final.

Структура дерева: токен, левое поддерево, правое поддерево.

Вектор нужен, чтобы при парсинге выражения хранить токены,

Если мы в дереве встречаем, что значение оператора = “/”, значение левого поддерева = “\*” и левого-левого поддерева равняется = “/” , то это перемножение дробей => сделать преобразование.

**7**. **Сценарий выполнения работы**

1.Создание структур.

2.Парсинг выражения.

3.Создание дерева выражения.

4.Преобразование над деревом выражений.

**8. Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми

примерами,подписанныйпреподавателем).

**TOKEN.H**

| //  // Created by machine on 30.04.2022.  //  #ifndef LAB24\_TOKEN\_H  #define LAB24\_TOKEN\_H  #include "stdbool.h"  typedef enum {  FINAL,  INTEGER,  FLOATING,  OPERATOR,  VARIABLE,  BRACKET  }TokenType;  typedef struct {  TokenType type;  union {  int value\_int;  float value\_float;  char operator\_name;  bool is\_left\_bracket;  char variable\_name;  } data;  } Token;  void token\_print(Token \*t, int depth);  void token\_next(Token \*t);  #endif //LAB24\_TOKEN\_H |
| --- |

**TREE.H**

| **//**  **// Created by machine on 30.04.2022.**  **//**  **#ifndef LAB24\_TREE\_H**  **#define LAB24\_TREE\_H**  **#include "token.h"**  **#include "malloc.h"**  **#include "vector.h"**  **typedef struct Tree{**  **Token data;**  **struct Tree\* left;**  **struct Tree\* right;**  **}Tree;**  **Tree\* tree\_create(vector\* v, int idx\_left, int idx\_right);**  **void tree\_print(Tree\* t, size\_t depth);**  **void tree\_infix(Tree\* t);**  **#endif //LAB24\_TREE\_H** |
| --- |

**VECTOR.H**

| **//**  **// Created by machine on 01.05.2022.**  **//**  **#ifndef LAB24\_VECTOR\_H**  **#define LAB24\_VECTOR\_H**  **#include "token.h"**  **#include <malloc.h>**  **#include <stdlib.h>**  **typedef struct {**  **int size;**  **int ptr;**  **Token\* t;**  **}vector;**  **void create(vector\* v);**  **void push(vector\* v, Token t);**  **void pop(vector \*v);**  **Token take(vector\*v, int idx);**  **#endif //LAB24\_VECTOR\_H** |
| --- |

**TRANSFORM.H**

| **//**  **// Created by machine on 01.05.2022.**  **//**  **#ifndef LAB24\_TRANSFORM\_H**  **#define LAB24\_TRANSFORM\_H**  **#include "tree.h"**  **void tree\_transform(Tree\* t);**  **#endif //LAB24\_TRANSFORM\_H** |
| --- |

**TREE.C**

| **//**  **// Created by machine on 30.04.2022.**  **//**  **#include "tree.h"**  **#include <stdlib.h>**  **#include <stdio.h>**  **int get\_priority(char c){**  **switch (c) {**  **case '+':**  **case '-':**  **return 1;**  **case '\*':**  **case '/':**  **return 2;**  **case '^':**  **return 3;**  **}**  **return 100;**  **}**  **Tree\* tree\_create(vector\* v, int idx\_left, int idx\_right){**  **Tree\* t = malloc(sizeof(Tree));**  **if(idx\_left > idx\_right){**  **return NULL;**  **}**  **if(idx\_right == idx\_left){**  **t->data = take(v, idx\_left);**  **t->left = NULL;**  **t->right = NULL;**  **return t;**  **}**  **int priority\_min = get\_priority('a');**  **int brackets = 0;**  **int priority;**  **int op\_pos;**  **for (int i = idx\_left; i < idx\_right; ++i) {**  **if (take(v, i).type == BRACKET && take(v, i).data.is\_left\_bracket) {**  **++brackets;**  **} else if (take(v, i).type == BRACKET && take(v, i).data.is\_left\_bracket == 0) {**  **--brackets;**  **} else if(brackets > 0){**  **continue;**  **} else if(take(v,i).type == OPERATOR){**  **priority = get\_priority(take(v,i).data.operator\_name);**  **if(priority <= priority\_min){**  **priority\_min = priority;**  **op\_pos = i;**  **}**  **}**  **}**  **if(priority\_min == 100 && take(v,idx\_left).type == BRACKET &&**  **take(v,idx\_left).data.is\_left\_bracket && take(v,idx\_right).type == BRACKET &&**  **!(take(v,idx\_right).data.is\_left\_bracket)){**  **free(t);**  **return tree\_create(v,idx\_left + 1, idx\_right-1);**  **}**  **if(take(v,op\_pos).data.operator\_name == '^'){**  **brackets=0;**  **for (int i = op\_pos; i >= idx\_left ; --i) {**  **if(take(v,i).type == BRACKET && take(v,i).data.is\_left\_bracket==0){**  **++brackets;**  **}**  **if (take(v,i).type == BRACKET && take(v,i).data.is\_left\_bracket) {**  **--brackets;**  **}**  **if(brackets > 0) continue;**  **if(take(v,i).type == OPERATOR){**  **priority = get\_priority(take(v,i).data.operator\_name);**  **if(priority == 3){**  **op\_pos = i;**  **}**  **}**  **}**  **}**  **t->data = take(v,op\_pos);**  **t->left = tree\_create(v, idx\_left, op\_pos-1);**  **t->right = tree\_create(v,op\_pos+1, idx\_right);**  **if(t->right == NULL){**  **printf("error");**  **exit(1);**  **}**  **return t;**  **}**  **void tree\_delete(Tree \*t){**  **if(t != NULL){**  **tree\_delete(t->left);**  **tree\_delete(t->right);**  **}**  **free(t);**  **}**  **void tree\_infix(Tree\* t){**  **if(t != NULL){**  **if(t->left && t->right) printf("(");**  **tree\_infix(t->left);**  **token\_print(&t->data,1);**  **tree\_infix(t->right);**  **if(t->right && t->left){**  **printf(")");**  **}**  **}**  **}**  **void tree\_print(Tree\* t, size\_t depth){**  **if(t != NULL){**  **tree\_print(t->left, depth + 10);**  **token\_print(&(t->data),depth);**  **printf("\n");**  **tree\_print(t->right, depth + 10);**  **}**  **}** |
| --- |

**VECTOR.C**

| **//**  **// Created by machine on 01.05.2022.**  **//**  **#include "vector.h"**  **void create(vector\* v){**  **v->ptr=0;**  **v->size = 200;**  **v->t = malloc(sizeof(Token)\* v->size);**  **}**  **void push(vector\* v, Token t){**  **if(v->size == v->ptr) {**  **v->size \*= 2;**  **v->t = realloc(v,sizeof(Token)\* v->size);**  **}**  **v->t[v->ptr++] = t;**  **}**  **void pop(vector\* v ){**  **v->ptr--;**  **}**  **Token take(vector\* v, int idx){**  **// if(idx >= v->ptr){**  **// exit(1);**  **// }**  **return v->t[idx];**  **}** |
| --- |

**TRANSFORM.C**

| **#include "token.h"**  **#include "token.c"**  **#include "transform.h"**  **int match\_frac(Tree \*t)**  **{**  **return (t->data.data.operator\_name == '/' && t->left->data.data.operator\_name == '\*' &&**  **t->left->left->data.data.operator\_name == '/');**  **}**  **void transform\_frac(Tree\* t)**  **{**  **Tree\* tmpright = malloc(sizeof(Tree));**  **Tree\* tmpleft = malloc(sizeof(Tree));**  **tmpright->data = t->left->data;**  **tmpright->left = t->left->left->right;**  **tmpright->right = t->right;**  **tmpleft->data = t->left->data;**  **tmpleft -> right = t->left->right;**  **tmpleft ->left = t->left->left->left;**  **t->left = tmpleft;**  **t->right = tmpright;**  **}**  **void tree\_transform(Tree \*t){**  **if (t != NULL) {**  **tree\_transform(t->left);**  **tree\_transform(t->right);**  **if (match\_frac(t)) {**  **transform\_frac(t);**  **}**  **}**  **}** |
| --- |

**TOKEN.C**

| **#include "token.h"**  **#include <stdlib.h>**  **#include <ctype.h>**  **#include <stdio.h>**  **void token\_next(Token\* t){**  **static bool mb\_unary = true;**  **char c;**  **do {**  **c = fgetc(stdin);**  **} while (isspace(c));**  **if (c == EOF) {**  **t->type = FINAL;**  **}**  **else if(isalpha(c) || c == '\_'){**  **mb\_unary = false;**  **t->type=VARIABLE;**  **t->data.variable\_name=c;**  **}**  **else if(isdigit(c)){**  **mb\_unary=false;**  **ungetc(c,stdin);**  **float f;**  **scanf("%f", &f);**  **if(f == (int)f){**  **t->type=INTEGER;**  **t->data.value\_int=(int)f;**  **} else {**  **t->type=FLOATING;**  **t->data.value\_float = f;**  **}**  **}**  **else if(c== '(' || c ==')'){**  **t->type=BRACKET;**  **if(c=='('){**  **mb\_unary = true;**  **t->data.is\_left\_bracket=1;**  **} else {**  **mb\_unary = false;**  **t->data.is\_left\_bracket = 0;**  **}**  **}**  **else if(mb\_unary && (c == '+' || c == '-')){**  **int znak = (c == '+') ? 1 : -1;**  **do {**  **c = fgetc(stdin);**  **} while (isspace(c));**  **if(isdigit(c)) {**  **ungetc(c, stdin);**  **token\_next(t);**  **if (t->type == INTEGER) {**  **t->data.value\_int = t->data.value\_int \* znak;**  **} else {**  **t->data.value\_float = t->data.value\_float \* znak;**  **}**  **} else {**  **ungetc(c,stdin);**  **t->type = OPERATOR;**  **t->data.operator\_name = '-';**  **mb\_unary = true;**  **}**  **}**  **else {**  **t->type = OPERATOR;**  **t->data.operator\_name = c;**  **mb\_unary = true;**  **}**  **}**  **void token\_print(Token\* t, int depth){**  **switch (t->type) {**  **case FINAL:**  **break;**  **case INTEGER:**  **printf("%\*d", depth, t->data.value\_int);**  **break;**  **case FLOATING:**  **printf("%\*f", depth, t->data.value\_float);**  **break;**  **case VARIABLE:**  **printf("%\*c", depth, t->data.variable\_name);**  **break;**  **case BRACKET:**  **printf("%\*c", depth, t->data.is\_left\_bracket == 1 ? '(' : ')');**  **break;**  **case OPERATOR:**  **printf("%\*c", depth, t->data.operator\_name);**  **break;**  **}**  **}** |
| --- |

**MAIN.C**

| **#include "vector.h"**  **#include "tree.h"**  **#include "token.h"**  **#include "transform.h"**  **#include "vector.c"**  **#include "tree.c"**  **#include "transform.c"**  **int main() {**  **vector\* tokens = malloc(sizeof(vector));**  **create(tokens);**  **size\_t tokens\_qty = 0;**  **Token token;**  **token\_next(&token);**  **while (token.type != FINAL){**  **push(tokens, token);**  **token\_next(&token);**  **tokens\_qty++;**  **}**  **Tree\* tree = tree\_create(tokens,0,(int)tokens\_qty-1);**  **printf("\nExpression tree:\n");**  **tree\_print(tree,0);**  **tree\_transform(tree);**  **printf("\nSemitransformed expression tree:\n");**  **tree\_print(tree,0);**  **return 0;**  **}** |
| --- |

**9. Дневник отладки**

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

1. **Замечания автора**
2. **Выводы**

Узнал, что такое дерево выражений и о множестве его реализаций. Реализовал инфиксное дерево выражений, вектор и токен дерева выражений. Узнал о таких функциях, как isdigit, isalpha, isspace.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_