**Отчет по лабораторной работе № 24** по курсу “ Практикум на ЭВМ ”

Студент группы М8О-102Б-21, Богуж Владислав Андреевич, № по списку 3

Контакты www, e-mail, icq, skype: boguzhvlad@gmail.com

Работа выполнена: « » \_\_\_\_\_\_\_\_\_\_\_\_201\_\_\_г.

Преподаватель: Никулин Сергей Петрович Каф.806\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » \_\_\_\_\_\_\_\_\_201 \_\_ г., итоговая оценка \_\_\_\_\_

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

1. **Тема:** Динамические структуры данных. Обработка деревьев.
2. **Цель работы:** Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев.
3. **Задание (вариант 19):** Вынести из произведений и делений унарные минусы
4. **Оборудование:**

*Оборудование ПЭВМ студента:*

Процессор: Apple M1, с ОЗУ 8 гб (виртуальная машина), SSD 256 гб. Монитор: встроенный 13 дюймов IPS 2560×1600, 220 PPI.

1. **Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства Linux, наименование Ubuntu, версия 22.04

Прикладные системы и программы: терминал OC UNIX, текстовый редактор emacs

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

1. Реализуем следующие структуры:

* Стек
* Дерево выражений

1. Реализуем следующие функции:

*Для дерева выражений*:

Node \*treeNodeCreate(void);

Node \*treeCopy(Node \*\*node);

Node \*treeMakeMinus(Node \*\*node);

Node \*treeMakeNotMinus(Node \*\*node);

int treeIsMinusNode(Node \*\*node);

void treeBuild(Node \*\*node, Stack \*st);

void treeDestroy(Node \*\*node);

void treeMoveMinus(Node \*\*node);

void treeCountMinus(Node \*\*node, int \*cnt);

void PKL(Node \*\*node, const int level);

void LKP(Node \*\*node);

*Для парсинга*:

int isLetter(const char ch);

int isNumber(const char ch);

int isOp(const char ch);

int isOpHigh(const char op1, const char op2);

void postOrder(const char \*str, Stack \*st);

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

Тесты:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Описание тестируемого случая** |
| 1\*(-2)\*2\*(-3) | ((1\*2)\*2)\*3 | Простое удаление унарного минуса |
| a\*b\*c | (a\*b)\*c | Выражение, не имеющее минусов |
| i\*(-k)\*j+u\*m\*(-d) | (-((i\*k)\*j))+(-((u\*m)\*d)) | Вынесение унарных минусов у отдельных произведений |
| 27\*(-2)/4\*(-4) | ((27\*2)/4)\*4 | Вынесение минусов при делении |

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

*Допущен к выполнению работы.* **Подпись преподавателя****\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем)

**Makefile**

CC = gcc

CFLAGS = -g -Wall -Wextra -pedantic -std=c99 -w -pipe -O3 -lm

lab24: stack.o lexer.o tree\_transform.o lab24.o

$(CC) $^ -o $@ $(CFLAGS)

clean:

rm -f \*.o lab24

**stack.h**

#ifndef STACK\_H

#define STACK\_H

#include <stdlib.h>

typedef struct \_Token {

char \_varOp;

int \_num;

} Token;

typedef Token STACK\_TYPE;

typedef struct \_ItemStack {

STACK\_TYPE \_data;

struct \_ItemStack \*\_prev;

} ItemStack;

typedef struct \_Stack {

int \_size;

struct \_ItemStack \*\_top;

} Stack;

void stackCreate(Stack \*s);

int stackEmpty(const Stack \*s);

int stackSize(const Stack \*s);

int stackPush(Stack \*s, const STACK\_TYPE value);

int stackPop(Stack \*s);

STACK\_TYPE stackTop(const Stack \*s);

void stackDestroy(Stack \*s);

#endif

**stack.c**

#include "stack.h"

void stackCreate(Stack \*s) {

s->\_size = 0;

s->\_top = NULL;

}

int stackEmpty(const Stack \*s) {

return s->\_top == NULL;

}

int stackSize(const Stack \*s) {

return s->\_size;

}

int stackPush(Stack \*s, const STACK\_TYPE value) {

ItemStack \*item = (ItemStack \*)malloc(sizeof(ItemStack));

if (!item)

return 0;

item->\_data = value;

item->\_prev = s->\_top;

s->\_top = item;

s->\_size++;

return 1;

}

int stackPop(Stack \*s) {

ItemStack \*item = NULL;

if (!s->\_size)

return 0;

item = s->\_top;

s->\_top = s->\_top->\_prev;

s->\_size--;

free(item);

return 1;

}

STACK\_TYPE stackTop(const Stack \*s) {

return s->\_top->\_data;

}

void stackDestroy(Stack \*s) {

ItemStack \*item = NULL;

while (s->\_top) {

item = s->\_top;

s->\_top = s->\_top->\_prev;

free(item);

}

s->\_size = 0;

s->\_top = NULL;

}

**lexer.h**

#ifndef LEXER\_H

#define LEXER\_H

#include <stdio.h>

#include <string.h>

int isLetter(const char ch);

int isNumber(const char ch);

int isOp(const char ch);

int isOpHigh(const char op1, const char op2);

void postOrder(const char \*str, Stack \*st);

#endif

**lexer.c**

#include <stdio.h>

#include <string.h>

int isLetter(const char ch) {

return ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'));

}

int isNumber(const char ch) {

return (ch >= '0' && ch <= '9');

}

int isOp(const char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^' || ch == '(' || ch == ')');

}

int opPrior(const char op) {

if (op == '^')

return 4;

if (op == '\*' || op == '/')

return 3;

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

return 2;

return 1;

}

int isOpHigh(const char op1, const char op2) {

if (op1 == '(' || op2 == '(' || op2 == ')')

return 0;

if (op1 == op2 && op2 == '^')

return 0;

return (opPrior(op1) >= opPrior(op2));

}

**tree\_transform.h**

#ifndef TREE\_TRANSFORM\_H

#define TREE\_TRANSFORM\_H

#include <stdio.h>

#include <string.h>

#include "stack.h"

typedef struct \_Node {

char \_varOp;

int \_num;

struct \_Node \*\_left;

struct \_Node \*\_right;

} Node;

Node \*treeNodeCreate(void);

Node \*treeCopy(Node \*\*node);

Node \*treeMakeMinus(Node \*\*node);

Node \*treeMakeNotMinus(Node \*\*node);

int treeIsMinusNode(Node \*\*node);

void treeBuild(Node \*\*node, Stack \*st);

void treeDestroy(Node \*\*node);

void treeMoveMinus(Node \*\*node);

void treeCountMinus(Node \*\*node, int \*cnt);

void PKL(Node \*\*node, const int level);

void LKP(Node \*\*node);

#endif

**tree\_transform.c**

#include <stdio.h>

#include <string.h>

#include "tree\_transform.h"

#include "lexer.h"

Node \*treeNodeCreate(void) {

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

tmpNode->\_varOp = '\0';

tmpNode->\_num = 0;

tmpNode->\_left = NULL;

tmpNode->\_right = NULL;

return tmpNode;

}

Node \*treeCopy(Node \*\*node) {

Node \*tmpNode = NULL;

if (\*node == NULL)

return NULL;

tmpNode = treeNodeCreate();

tmpNode->\_varOp = (\*node)->\_varOp;

tmpNode->\_num = (\*node)->\_num;

tmpNode->\_left = treeCopy(&(\*node)->\_left);

tmpNode->\_right = treeCopy(&(\*node)->\_right);

return tmpNode;

}

Node \*treeMakeMinus(Node \*\*node) {

Node \*tmpNode = NULL;

tmpNode = treeNodeCreate();

tmpNode->\_left = treeNodeCreate();

tmpNode->\_right = treeCopy(node);

tmpNode->\_varOp = '-';

free(\*node);

\*node = NULL;

return tmpNode;

}

Node \*treeMakeNotMinus(Node \*\*node) {

Node \*tmpNode = treeCopy(&(\*node)->\_right);

treeDestroy(node);

return tmpNode;

}

int treeIsMinusNode(Node \*\*node) {

if (\*node == NULL)

return 0;

if ((\*node)->\_left == NULL || (\*node)->\_right == NULL)

return 0;

return ((\*node)->\_varOp == '-' && (\*node)->\_left->\_varOp == '\0' && (\*node)->\_left->\_num == 0);

}

void treeBuild(Node \*\*node, Stack \*st) {

Token token;

if (stackEmpty(st))

return;

token = stackTop(st);

stackPop(st);

(\*node) = treeNodeCreate();

(\*node)->\_varOp = token.\_varOp;

(\*node)->\_num = token.\_num;

if (isOp((\*node)->\_varOp)) {

treeBuild(&(\*node)->\_right, st);

treeBuild(&(\*node)->\_left, st);

}

}

void treeDestroy(Node \*\*node) {

if (\*node == NULL)

return;

if ((\*node)->\_left != NULL)

treeDestroy(&(\*node)->\_left);

if ((\*node)->\_right != NULL)

treeDestroy(&(\*node)->\_right);

free(\*node);

\*node = NULL;

}

void treeMoveMinus(Node \*\*node) {

int minusCnt = 0;

if (\*node == NULL)

return;

if ((\*node)->\_varOp == '\*' || (\*node)->\_varOp == '/'){

treeCountMinus(node, &minusCnt);

if (minusCnt & 1)

\*node = treeMakeMinus(node);

} else if (isOp((\*node)->\_varOp)) {

treeMoveMinus(&(\*node)->\_left);

treeMoveMinus(&(\*node)->\_right);

}

}

void treeCountMinus(Node \*\*node, int \*cnt) {

if (\*node == NULL)

return;

if (treeIsMinusNode(node)) {

\*node = treeMakeNotMinus(node);

(\*cnt)++;

} else if ((\*node)->\_varOp == '\*' || (\*node)->\_varOp == '/') {

treeCountMinus(&(\*node)->\_left, cnt);

treeCountMinus(&(\*node)->\_right, cnt);

} else {

treeMoveMinus(&(\*node)->\_left);

treeMoveMinus(&(\*node)->\_right);

}

}

void PKL(Node \*\*node, const int level) {

if (\*node == NULL)

return;

if ((\*node)->\_right != NULL)

PKL(&(\*node)->\_right, level + 1);

if ((\*node)->\_varOp != '\0')

printf("%\*s%c\n", level \* 4, "", (\*node)->\_varOp);

else

printf("%\*s%d\n", level \* 4, "", (\*node)->\_num);

if ((\*node)->\_left != NULL)

PKL(&(\*node)->\_left, level + 1);

}

void LKP(Node \*\*node) {

if (\*node == NULL)

return;

if ((\*node)->\_left != NULL && !treeIsMinusNode(node)) {

if ((\*node)->\_left->\_left != NULL)

printf("(");

LKP(&(\*node)->\_left);

if ((\*node)->\_left->\_left != NULL)

printf(")");

}

if ((\*node)->\_varOp != '\0')

printf("%c", (\*node)->\_varOp);

else

printf("%d", (\*node)->\_num);

if ((\*node)->\_right != NULL) {

if ((\*node)->\_right->\_left != NULL)

printf("(");

LKP(&(\*node)->\_right);

if ((\*node)->\_right->\_left != NULL)

printf(")");

}

}

void postOrder(const char \*str, Stack \*st)

{

int i = 0, isBracket = 0;

char tmpCh;

Token tk;

Stack stOp;

stackCreate(&stOp);

tk.\_varOp = '\0';

tk.\_num = 0;

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

if (isLetter(str[i])) {

tk.\_varOp = str[i];

stackPush(st, tk);

} else if (isNumber(str[i])) {

tk.\_varOp = '\0';

tk.\_num = tk.\_num \* 10 + str[i] - '0';

if (str[i + 1] != '.' && !isNumber(str[i + 1])) {

stackPush(st, tk);

tk.\_num = 0;

}

}

else if (isOp(str[i])) {

tk.\_varOp = str[i];

if (str[i] == ')')

isBracket = 1;

else if (str[i] == '-' && (i == 0 || str[i - 1] == '(')) {

tmpCh = tk.\_varOp;

tk.\_varOp = '\0';

tk.\_num = 0;

stackPush(st, tk);

tk.\_varOp = tmpCh;

}

while (!stackEmpty(&stOp) && (isOpHigh(stackTop(&stOp).\_varOp, str[i]) || isBracket)) {

if (stackTop(&stOp).\_varOp == '(')

isBracket = 0;

else

stackPush(st, stackTop(&stOp));

stackPop(&stOp);

}

if (str[i] != ')')

stackPush(&stOp, tk);

}

i++;

}

while (!stackEmpty(&stOp)) {

stackPush(st, stackTop(&stOp));

stackPop(&stOp);

}

stackDestroy(&stOp);

}

**lab24.c**

#include <stdio.h>

#include <string.h>

#include "stack.h"

#include "tree\_transform.h"

#include "lexer.h"

void menu() {

printf("1) Ввести выражение\n");

printf("2) Вывести исходное выражение\n");

printf("3) Вывести преобразованное выражение\n");

printf("4) Вывести исходное дерево\n");

printf("5) Вывести преобразованное дерево\n");

printf("6) Меню\n");

printf("7) Выход\n");

}

int main(void) {

int action;

char expr[255];

Node \*root = NULL, \*root2 = NULL;

Stack stPost;

printf("\n║Меню:\n");

printf("1) Ввести выражение\n");

printf("2) Вывести исходное выражение\n");

printf("3) Вывести преобразованное выражение\n");

printf("4) Вывести исходное дерево\n");

printf("5) Вывести преобразованное дерево\n");

printf("6) Меню\n");

printf("7) Выход\n");

while (1) {

printf("\nВыберите действие из списка ");

scanf("%d", &action);

switch (action){

case 1:

printf("\n║Введите выражение: ");

scanf("%s", expr);

treeDestroy(&root);

treeDestroy(&root2);

stackCreate(&stPost);

postOrder(expr, &stPost);

treeBuild(&root, &stPost);

stackDestroy(&stPost);

root2 = treeCopy(&root);

treeMoveMinus(&root2);

break;

case 2:

printf("\n║Исходное выражение: %s\n", expr);

break;

case 3:

printf("\n║Преобразованное выражение: ");

LKP(&root2);

printf("\n");

break;

case 4:

if (root != NULL) {

printf("\n║Дерево исходное выражения\n");

PKL(&root, 0);

}

else

printf("\n");

break;

case 5:

if (root2 != NULL) {

printf("\n║Дерево преобразованного выражения\n");

PKL(&root2, 0);

} else {

printf("\n");

}

break;

case 6:

printf("\n║Меню:\n");

menu();

break;

case 7:

return 0;

break;

default:

printf("\nОшибка. Действия с таким номером не существует, выберите команду из меню:\n");

menu();

break;

}

}

treeDestroy(&root);

treeDestroy(&root2);

return 0;

}

**Работа программы:**

**║Меню:**

**1) Ввести выражение**

**2) Вывести исходное выражение**

**3) Вывести преобразованное выражение**

**4) Вывести исходное дерево**

**5) Вывести преобразованное дерево**

**6) Меню**

**7) Выход**

**Выберите действие из списка 1**

**║Введите выражение: 1\*(-2)\*2\*(-3)**

**Выберите действие из списка 2**

**║Исходное выражение: 1\*(-2)\*2\*(-3)**

**Выберите действие из списка 3**

**║Преобразованное выражение: ((1\*2)\*2)\*3**

**Выберите действие из списка 4**

**║Дерево исходное выражения**

**3**

**-**

**0**

**\***

**2**

**\***

**2**

**-**

**0**

**\***

**1**

**Выберите действие из списка 5**

**║Дерево преобразованного выражения**

**3**

**\***

**2**

**\***

**2**

**\***

**1**

**Выберите действие из списка 6**

**║Меню:**

**1) Ввести выражение**

**2) Вывести исходное выражение**

**3) Вывести преобразованное выражение**

**4) Вывести исходное дерево**

**5) Вывести преобразованное дерево**

**6) Меню**

**7) Выход**

**Выберите действие из списка 1**

**║Введите выражение: a\*b\*c**

**Выберите действие из списка 2**

**║Исходное выражение: a\*b\*c**

**Выберите действие из списка 3**

**║Преобразованное выражение: (a\*b)\*c**

**Выберите действие из списка 4**

**║Дерево исходное выражения**

**c**

**\***

**b**

**\***

**a**

**Выберите действие из списка 5**

**║Дерево преобразованного выражения**

**c**

**\***

**b**

**\***

**a**

**Выберите действие из списка 6**

**║Меню:**

**1) Ввести выражение**

**2) Вывести исходное выражение**

**3) Вывести преобразованное выражение**

**4) Вывести исходное дерево**

**5) Вывести преобразованное дерево**

**6) Меню**

**7) Выход**

**Выберите действие из списка 1**

**║Введите выражение: i\*(-k)\*j+u\*m\*(-d)**

**Выберите действие из списка 2**

**║Исходное выражение: i\*(-k)\*j+u\*m\*(-d)**

**Выберите действие из списка 3**

**║Преобразованное выражение: (-((i\*k)\*j))+(-((u\*m)\*d))**

**Выберите действие из списка 4**

**║Дерево исходное выражения**

**d**

**-**

**0**

**\***

**m**

**\***

**u**

**+**

**j**

**\***

**k**

**-**

**0**

**\***

**i**

**Выберите действие из списка 5**

**║Дерево преобразованного выражения**

**d**

**\***

**m**

**\***

**u**

**-**

**0**

**+**

**j**

**\***

**k**

**\***

**i**

**-**

**0**

**Выберите действие из списка 6**

**║Меню:**

**1) Ввести выражение**

**2) Вывести исходное выражение**

**3) Вывести преобразованное выражение**

**4) Вывести исходное дерево**

**5) Вывести преобразованное дерево**

**6) Меню**

**7) Выход**

**Выберите действие из списка 1**

**║Введите выражение: 27\*(-2)/4\*(-4)**

**Выберите действие из списка 2**

**║Исходное выражение: 27\*(-2)/4\*(-4)**

**Выберите действие из списка 3**

**║Преобразованное выражение: ((27\*2)/4)\*4**

**Выберите действие из списка 4**

**║Дерево исходное выражения**

**4**

**-**

**0**

**\***

**4**

**/**

**2**

**-**

**0**

**\***

**27**

**Выберите действие из списка 5**

**║Дерево преобразованного выражения**

**4**

**\***

**4**

**/**

**2**

**\***

**27**

**Выберите действие из списка 6**

**║Меню:**

**1) Ввести выражение**

**2) Вывести исходное выражение**

**3) Вывести преобразованное выражение**

**4) Вывести исходное дерево**

**5) Вывести преобразованное дерево**

**6) Меню**

**7) Выход**

**Выберите действие из списка 7**

1. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные события(ошибки в сценарии и программе,нестандартные ситуации) 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. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы: -
2. **Выводы:**

Я научился работать с динамическими структурами и обрабатывать деревья выражений

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