**Отчет по лабораторной работе № 24** по курсу “Языки и методы программирования”

Студент группы М80-103Б-21 Фадеев Денис Вадимович, № по списку 22

Контакты e-mail: denfad2003@mail.ru, telegram: @Denissimo\_f

Работа выполнена: «19» мая 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

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

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

* 1. **Тема: деревья арифметических выражений.**
  2. **Цель работы:** Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев.
  3. **Задание:** вариант №9 Умножение переменной на сумму в скобках заменить на сумму произведений
  4. **Оборудование** :

Процессор *Intel Core i5-6200U @ 4x 2.30GHz* с ОП 16 Гб, НМД 512 Гб. Монитор *1920x1080*

**5. Программное обеспечение.**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия 20*.04 focal*

интерпретатор команд: *bash* версия 5.0.16(1)-release.

Редактор текстов *emacs* версия 26.3

**6. Идея, метод, алгоритм**

Необходимо создать структуру данных и функции её обработки. Алгоритм должен будет из вводимого выражения создавать дерево выражения с учётом скобок. После чего нужно провести преобразование согласно заданию и вывести вид дерева и итоговое выражение.

**7.Сценарий выполнения работы**

Создать файл tree.c и tree.h со следующими реализованными функциями:

* define\_priority - возвращает приоритет действия
* make\_tree - создаёт дерево из строчного выражения
* print\_tree - выводит дерево выражения
* print\_expression - выводит выражение из дерева
* copy - копирует дерево
* spec\_func - специальная функция выполняющая действие из задания

Выполнение программы:

**Choose an action:**

**1. Enter an expression and create tree.**

**2. Transforn expression.**

**3. Print tree.**

**4. Print expression.**

**5. Exit**

**1**

**Please, enter an expression: (a+b)\*c+d\*(e+c)**

**Choose an action:**

**1. Enter an expression and create tree.**

**2. Transforn expression.**

**3. Print tree.**

**4. Print expression.**

**5. Exit**

**2**

**Choose an action:**

**1. Enter an expression and create tree.**

**2. Transforn expression.**

**3. Print tree.**

**4. Print expression.**

**5. Exit**

**3**

**c**

**\***

**d**

**+**

**e**

**\***

**d**

**+**

**b**

**\***

**c**

**+**

**a**

**\***

**c**

**Choose an action:**

**1. Enter an expression and create tree.**

**2. Transforn expression.**

**3. Print tree.**

**4. Print expression.**

**5. Exit**

**4**

**c\*a+c\*b+d\*e+d\*c**

**8. Распечатка протокола**

**tree.c**

#include "tree.h"

bool is\_op(char a) {

if(a == '\*' || a == '^' || a == '/' || a == '+' || a == '-') return true;

**else return false;**

}

int define\_priority (char a) {

switch (a){

case '-':

case '+':

return 1;

case '\*':

return 2;

case '/':

return 3;

case '^':

return 4;

default:

return 100;

}

}

unit \* make\_tree(char expr[], int first, int last) {

int prior, min\_prior = 100, k, depth = 0;

unit \*tree = malloc(sizeof(unit));

for (int i = first; i <= last; ++i) {

if (expr[i] == '(') {

depth++;

continue;

}

if (expr[i] == ')') {

depth--;

continue;

}

if (depth>0){

continue;

}

prior = define\_priority(expr[i]);

if (prior <= min\_prior) {

min\_prior = prior;

k = i;

}

}

if (depth !=0) {

printf("Wrong expression!\n");

exit (1);

}

int l;

if (min\_prior == 100) {

if (expr[first] == '(' && expr[last] == ')') {

free(tree);

return make\_tree(expr, first +1, last - 1);

}

else {

l = last - first + 1;

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

tree->data[i] = expr[first+i];

}

tree->data[l] = '\n';

tree->left = NULL;

tree->right = NULL;

return tree;

}

}

tree->data[0] = expr[k];

tree->data[1] = '\n';

tree->left = make\_tree(expr, first, k-1);

tree->right = make\_tree(expr, k+1, last);

return tree;

}

void print\_tree(unit \*u, int l) {

if (u->right != NULL) print\_tree(u->right, l+1);

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

printf(" ");

}

printf("%5s", u->data);

if (u->left != NULL) print\_tree(u->left, l+1);

}

void print\_expression(unit \*u) {

if (u==NULL) {

return;

}

if (define\_priority(u->data[0])!=100 && define\_priority(u->left->data[0])!=100 && define\_priority(u->data[0])

> define\_priority(u->left->data[0]) || u->data[0] == '^' && u->left->data[0] == '^' ){

printf("(");

print\_expression(u->left);

printf(")");

} else print\_expression(u->left);

for (int i = 0; i < 50; ++i) {

if (u->data[i] == '\n') {

break;

}

printf("%c", u->data[i]);

}

if (define\_priority(u->data[0])!=100 && define\_priority(u->right->data[0])!=100 && define\_priority(u->data[0]

) > define\_priority(u->right->data[0]) || u->data[0] == '^' && u->right->data[0] == '^' ){

printf("(");

print\_expression(u->right);

printf(")");

} else print\_expression(u->right);

}

unit \* copy(unit \*u) {

if (u == NULL) {

return NULL;

}

unit \*f = malloc(sizeof(unit));

for (int i = 0; i<50; ++i) {

f->data[i] = u->data[i];

}

f ->left = copy(u->left);

f->right = copy(u->right);

return f;

}

unit \* spec\_func(unit \*u) {

if (u == NULL) {

return NULL;

}

if(is\_op(u->data[0])) {

spec\_func(u->left);

spec\_func(u->right);

}

if(u->data[0] == '\*' && !is\_op(u->left->data[0]) && u->right->data[0] == '+'){

u->data[0] = '+';

unit \*left = u->left;

unit \*right = u->right;

u->left = make\_tree("\*", 0, 1);

u->left->left = left;

u->left->right = right->left;

u->right = make\_tree("\*", 0, 1);

u->right->left = left;

u->right->right = right->right;

}

else if(u->data[0] == '\*' && !is\_op(u->right->data[0]) && u->left->data[0] == '+'){

u->data[0] = '+';

unit \*left = u->left;

unit \*right = u->right;

u->left = make\_tree("\*", 0, 1);

u->left->left = right;

u->left->right = left->left;

u->right = make\_tree("\*", 0, 1);

u->right->left = right;

u->right->right = left->right;

}

**return u;**

}

**tree.h**

#ifndef tree\_h

#define tree\_h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

typedef struct unit {

char data [50];

struct unit \*left;

struct unit \*right;

}unit;

int define\_priority (char a);

unit \* make\_tree(char expr[], int first, int last);

void print\_tree(unit \*u, int l);

void print\_expression(unit \*u);

unit \* copy(unit \*u);

unit \* spec\_func(unit \*u);

#endif /\* tree\_h \*/

**main.c**

#include "tree.h"

int main(void) {

unit \*t = NULL;

int opt = -1;

while (opt!=5) {

printf("Choose code of action:\n1. Enter an expression and create tree.\t2. Transforn expression.\n3. Print tree.\tt4. Print expression.\t5. Exit\n");

scanf("%d", &opt);

switch (opt) {

case 1: {

printf("Please, enter an expression: ");

char expression[1000];

scanf("%s", expression);

int n = 0;

while (expression[n] != '\0') {

n++;

}

t = make\_tree(expression, 0, n-1);

break;

}

case 2: {

t = spec\_func(t);

break;

}

case 3: {

printf("\n");

print\_tree(t, 0);

break;

}

case 4: {

printf("\n");

print\_expression(t);

printf("\n");

break;

}

}

}

return 0;

}

**9. Дневник отладки**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. Или Дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | - | - | - | - | - | - |

1. **Замечания автора** -
2. **Выводы**

Результатом лабораторной работы стало создание программы с алгоритмом, позволяющим строить деревья выражений и обрабатывать их. Самым сложным было создать алгоритм построения дерева, так как он содержит много кода и рекурсивных вызовов. Но в целом выполнение лабораторной работы мне понравилось, она позволила мне лучше разобраться в работе калькуляторов и возможно это поможет в будущем.

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