

Отчет по лабораторной работе № 24 по курсу «Практикум на ЭВМ»

Студент группы Алапанова Эльза Халилевна, № по списку 3

Контакты e-mail : alapanowa02@yandex.ru

Работа выполнена: «25» мая 2021г.

Преподаватель: каф. 806 Найденков Иван
Евгеньевич

Отчет сдан « » _____ 20 ____ г.,
итоговая оценка _____

Подпись преподавателя

1. Тема: Преобразование выражения в дерево.

2. Цель работы: Научиться преобразовывать выражения в деревья и работать с ними.

3. Задание (вариант № 23): Упростить дробь, сократив в числителе и знаменателе общие переменные и константы.

4. Оборудование (студенческое)

Процессор Intel® Core™ i5-10210 @ 1.60 GHz с ОП 8192 Мб, НМД 512 Уб.

Монитор 1920 x 1080

5. Программное обеспечение (студенческое):

Операционная система семейства Ubuntu, наименование Ubuntu 20.04.2 LTS
версия _____

интерпретатор команд _____ версия _____.

Система программирования _____

_____ версия _____

Редактор текстов _____

_____ версия _____

Утилиты операционной системы _____

Прикладные системы и программы Sublime Text

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

Опишем структуру дерева выражения:

- struct tnode {
 char data[40];
 struct tnode *left, *right;
};
typedef struct tnode *node;

Структура хранит массив символов, а также указатели на левого и правого потомка

С помощью этой структуры реализуем функции для работы с деревом выражения:

- int op(char data[40])
Функция определения операции. Возвращает 1, если это операция и 0 в противном случае
- int priority(char c)
Функция выдачи приоритета. Приоритет суммы и разности равен 1, умножения и деления — 2, возведения в степень — 3. В остальных случаях приоритет 100.
- node MakeTree(char Expr[], int first, int last)
Функция создания дерева выражений. Получает на вход выражение, а также индексы, которые необходимо обработать. Сначала создает пустое дерево, а затем находит операцию с самым минимальным приоритетом, пропуская операции в скобках, попутно считая количество открытых скобок. Если после обхода всего выражения количество открытых скобок не равно 0, то программа экстренно завершается. Если минимальная операция не найдена (значение равно 100), то считываем операнд. Если операция найдена, то помещаем ее в значение узла дерева, а затем рекурсивно находим левый и правый операнды
- node parent(node t, node son)
Функция поиска родителя по узлу дерева. Если текущий узел (t) равен искомому, то функция возвращает текущий узел. В противном случае функция вызывает рекурсию от левого поддерева и правого поддерева. Затем возвращает левый узел, если он не равен NULL, в противном случае — правый.
- void print_expression(node t)
Функция печати выражения по дереву. Если левое поддерево является операцией, чей приоритет ниже текущего (или обе операции - возведение в степень), то ставится открывающая скобка, затем выводится левое поддерево, затем ставится закрывающая скобка. В противном случае просто выводится левое поддерево. Затем выводится текущий узел. Далее все повторяется для правого поддерева.
- void transform(node T, node t)
Функция сокращения при помощи частых случаев.

В основной части программы будем использовать меню, в котором есть 5 опций:

1. Создание дерева (Create tree)
Запрашивает выражение, затем считывает его и создает дерево t путем вызова функции MakeTree
2. Трансформация выражения (Transform expression)
Вызывает функцию transform
3. Печать дерева (Print tree)
Вызывает функцию print_tree

4. Печать выражения (Print expression)
Вызывает функцию print_expression
5. Выход (Exit)
Выходит из меню

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

Тесты:

- 1) $(a*b)/a$
- 2) $(3*a)/(3*b)$
- 3) $(a*b)/(c*d)$
- 4) $(a*b*3)/(3*b*c)$
- 5) $a/(a*b)$
- 6) $a/(b*a)$

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

8. Распечатка протокола (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

elza@elza-NBLB-WAX9N:~/24\$ cat laba24.c

```
#include <stdio.h>
#include <stdlib.h>

struct tnode {
    char data[40];
    struct tnode *left, *right;
};
typedef struct tnode *node;
int op(char data[40]) {
    if (data[0] == '^' || data[0] == '/' || data[0] == '*' || data[0] == '+' || data[0] == '-') {
        return 1;
    }
    return 0;
}
int priority(char c) {
    switch (c) {
        case '-':
        case '+':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 100;
    }
}
node MakeTree(char Expr[], int first, int last) {
    int MinPrt, k, prt, nest = 0;
    node Tree = (node) malloc(sizeof(node));
    MinPrt = 100;
    for (int i = first; i <= last; i++) {
        if (Expr[i] == '(') {
            nest++;
            continue;
        }
        if (Expr[i] == ')') {
            nest--;
            continue;
        }
        if (nest > 0) {
            continue;
        }
        prt = priority(Expr[i]);
        if (prt <= MinPrt) {
            MinPrt = prt;
            k = i;
        }
    }
    if (nest != 0) {
        printf("Wrong expression\n");
        exit(1);
    }
    if (MinPrt == 100) {
        if (Expr[first] == '(' && Expr[last] == ')') {
            free(Tree);
            return MakeTree(Expr, first + 1, last - 1);
        } else {
            k = last - first + 1;

            for (int i = 0; i < k; i++) {
                Tree->data[i] = Expr[first + i];
            }
            Tree->data[k] = '\n';
            Tree->left = NULL;
            Tree->right = NULL;
            return Tree;
        }
    }
}
```

```

Tree->data[0] = Expr[k];
Tree->data[1] = '\n';
Tree->left = MakeTree(Expr, first, k - 1);
Tree->right = MakeTree(Expr, k + 1, last);
return Tree;
}
node parent(node t, node son) {
    if (t == NULL) {
        return NULL;
    }
    if (t->left == son || t->right == son) {
        return t;
    }
    node left = parent(t->left, son);
    node right = parent(t->right, son);
    return (left == NULL) ? right : left;
}
void print_tree(node t, int x) {
    if (t == NULL) {
        return;
    }
    print_tree(t->right, x + 1);
    for (int i = 0; i < x; i++) {
        printf("\t");
    }
    for (int i = 0; i <= 40; i++) {
        if (t->data[i] != '\n') {
            printf("%c", t->data[i]);
            continue;
        }
        printf("\n");
        break;
    }
    print_tree(t->left, x + 1);
}
void print_expression(node t) {
    if (t == NULL) {
        return;
    }
    if (op(t->data) && op(t->left->data) &&
        (priority(t->left->data[0]) < priority(t->data[0]) || t->left->data[0] == '^' && t->data[0] == '^')) {
        printf("(");
        print_expression(t->left);
        printf(")");
    } else {
        print_expression(t->left);
    }
    for (int i = 0; i <= 40; i++) {
        if (t->data[i] != '\n') {
            printf("%c", t->data[i]);
            continue;
        }
        break;
    }
    if (op(t->data) && op(t->right->data) &&
        (priority(t->right->data[0]) < priority(t->data[0]) || t->right->data[0] == '^' && t->data[0] == '^')) {
        printf("(");
        print_expression(t->right);
        printf(")");
    } else {
        print_expression(t->right);
    }
}
void transform(node T, node t) {
    if (t == NULL) {
        return;
    }
    if (t->data[0] == '/') {
        if (t->right->data[0] == '*' && t->left->data[0] == '*'){
            if (t->right->right->data[0] == t->left->right->data[0]){
                t->right->right->data[0] = '1';
                t->left->right->data[0] = '1';
            }
            if (t->right->right->data[0] == t->left->left->data[0]){

```

```

t->right->right->data[0] = '1';
t->left->left->data[0] = '1';
}
if (t->right->left->data[0] == t->left->right->data[0]){
t->right->left->data[0] = '1';
t->left->right->data[0] = '1';
}
if (t->right->left->data[0] == t->left->left->data[0]){
t->right->left->data[0] = '1';
t->left->left->data[0] = '1';
}
}
if (t->right->data[0] != t->left->data[0] && t->left->data[0] == '*'){
t->data[0] = '/';
if (t->right->data[0] == t->left->right->data[0]){
t->right->data[0] = '1';
t->left->right->data[0] = '1';
}
if (t->right->data[0] == t->left->left->data[0]){
t->right->data[0] = '1';
t->left->left->data[0] = '1';
}
}
}
if (t->right->data[0] != t->left->data[0] && t->right->data[0] == '*'){
t->data[0] = '/';
if (t->left->data[0] == t->right->right->data[0]){
t->left->data[0] = '1';
t->right->right->data[0] = '1';
}
if (t->left->data[0] == t->right->left->data[0]){
t->left->data[0] = '1';
t->right->left->data[0] = '1';
}
}
}
if (t->right->data[0] == '*' && t->right->right->data[0] == '1'){
node a = t->right->left->data[0];
t->right->data[0] = a;
t->right->left->data[0] = NULL;
t->right->right->data[0] = NULL;
printf("4\n");
}
if (t->right->data[0] == '*' && t->right->left->data[0] == '1'){
node a = t->right->right->data[0];
t->right->data[0] = a;
t->right->left->data[0] = NULL;
t->right->right->data[0] = NULL;
}
if (t->right->data[0] == '1' && t->left->data[0] == '*') {
node a = t->left->left;
node b = t->left->right;
t->data[0] = '*';
t->left = a;
t->right = b;
}
if (t->right->data[0] == t->left->data[0] && t->left->data[0] != '*' && t->right->data[0] != '*'){
t->data[0] = '1';
t->left->data[0] = NULL;
t->right->data[0] = NULL;
}
if (t->right->data[0] == '1' && t->left->data[0] != '*') {
node a = t->left->data[0];
t->data[0] = a;
t->left->data[0] = NULL;
t->right->data[0] = NULL;
}
if (t->data[0] == '*' && t->right->data[0] == '1'){
node a = t->left->data[0];
t->data[0] = a;
t->left->data[0] = NULL;
t->right->data[0] = NULL;
}
if (t->data[0] == '*' && t->left->data[0] == '1'){
node a = t->right->data[0];
t->data[0] = a;
t->left->data[0] = NULL;
t->right->data[0] = NULL;
}

```

```

    }
    if (t->left->data[0] == '*' && (t->left->right->data[0] == '1' || t->left->left->data[0] == '1')){
        if (t->left->right->data[0] == '1'){
            node a = t->left->left->data[0];
            t->left->data[0] = a;
        }
        else {
            node a = t->left->right->data[0];
            t->left->data[0] = a;
        }
        t->left->right->data[0] = NULL;
        t->left->left->data[0] = NULL;
    }
}
}
int main() {
    node t = NULL;
    int choose, g = 1;
    while (g) {
        printf("1. Create tree\t 2. Transform expression\t 3. Print tree\t 4. Print expression\t 5.Exit\n");
        scanf("%d", &choose);
        switch (choose) {
            case 1: {
                printf("Input your expression\n");
                char Expr[1000];
                scanf("%s", Expr);
                int n = 0;

                while (Expr[n] != '\0') {
                    n++;
                }
                t = MakeTree(Expr, 0, n - 1);
                parent(t, NULL);
                break;
            }
            case 2: {
                transform(t, t);
                break;
            }
            case 3: {
                print_tree(t, 0);
                break;
            }
            case 4: {
                print_expression(t);
                printf("\n");
                break;
            }
            case 5: {
                g = 0;
                break;
            }
            default: {
                printf("Wrong answer\n");
            }
        }
    }
    return 0;
}

```

elza@elza-NBLB-WAX9N:~/24\$ gcc laba24.c

elza@elza-NBLB-WAX9N:~/24\$./a.out

1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit

1

Input your expression

(a*b)/a

1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit

3

a

/

b

*

a

1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit

2

1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit

4
b
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
1
Input your expression
(3*a)/(3*b)
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
3
b
*
3
/
a
*
3
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
2
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
3
b
/
a
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
4
a/b
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
1
Input your expression
(a*b)/(c*d)
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
3
d
*
c
/
b
*
a
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
2
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
3
d
*
c
/
b
*
a
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
4
a*b/c*d
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
1
Input your expression
(a*b*3)/(3*b*c)
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
3
c
*
b
*
3
/
3
*
b
*
a
1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
2

5
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 3
 c
 /
 a
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 4
 a/c
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 1
 Input your expression
 a/(a*b)
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 3
 b
 *
 a
 /
 a
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 2
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 3
 b
 /
 1
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 4
 1/b
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 1
 Input your expression
 a/(b*a)
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 3
 a
 *
 b
 /
 a
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 2
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 3
 b
 /
 1
 1. Create tree 2. Transform expression 3. Print tree 4. Print expression 5.Exit
 4
 1/b

9. 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						
2						

10. Замечания автора по существу работы : замечаний нет.

11. Выводы : Научилась преобразовывать деревья в выражения и работать с ними.
Недочёты при выполнении задания могут быть устранены следующим образом:

_____ Подпись
студента _____