Отчёт по лабораторной работе № 24 по курсу Прикладная информатика

Студент группы М8О-104Б-22 Мазаев Василий Владимирович, № по списку 10

Контакты www, e-mail, icq, skype vm1717v@gmail.com

Работа выполнена: «21» мая 20 23 г.

Преподаватель: Потенко М. А. каф.806 _____

Входной контроль знаний с оценкой _____

Отчёт сдан « » _____201 __ г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема:** Дерево выражений на языке Си

2. **Цель работы:** составить программу выполнения заданных преобразований арифметических выражений с применением деревьев на языке Си.

3. **Задание** (*вариант №* 26): Заменить степень с разностью в показателе на частное степеней

4. **Оборудование(лабораторное)**:
ЭВМ _____, процессор _____, имя узла сети_____ с ОП _____ Мб,
НМД_____Мб. Терминал_____адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор AMD Ryzen 5 5500U with Radeon Graphics ОП 8192 Мб, НМД 524 288 Мб. Монитор _____
Другие устройства _____

5. **Программное обеспечение(лабораторное):**
Операционная система семейства _____, наименование _____ версия _____
интерпретатор команд _____ версия _____
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы_____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства UNIX, наименование Ubuntu версия 18.04.6
интерпретатор команд bash версия 18.04.6.
Система программирования _____версия _____
Редактор текстов nano версия _____
Утилиты операционной системы Терминал
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных на домашнем компьютере /Users/diss/24/main.c
Users/diss/23/mytree.c; Users/diss/23/mytree.h

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

Переменные или числа должны располагаться в листах дерева, операции - в узлах. Для начала преобразуем выражение в "обратную польскую нотацию" используя стек: посимвольно считывая выражение будем определять что в данный момент мы кладём в стек. В узлах стека будет храниться информация о типе узла (переменная, число, операция и т.п) и о значении узла в удобном для обработки виде (при выводе мы всё декодируем). Далее будем брать вершину стека и класть последовательно в дерево, таким образом, мы построим дерево выражений. Для решения задания нам необходимо будет разбить узел "^", который в правом потомке содержит разность, на два узла - частное степеней.

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

Для работы я заведу три файла: main.c (главный код программы и интерфейс работы с пользователем), mytree.c
(функции над стеком и деревом), mytree.h (обозначение самого стека, дерева и функций над ними).

Сначала обозначим структуру Node с помощью "typedef": в ней будут атрибуты - type (тип узла: переменная/число/операция), val, ch (значение узла), rang (приоритет операции), l, r (указатели на потомков). На этой структуре будет базироваться наш стек stack (указатель на следующий элемент только l) и дерево tree (с атрибутом root - корень дерева).

Функции над стеком и деревом запишем в файл mytree.c: пропишем базовые функции создания стека и дерева,
вставки элемента в стек, взятия и удаления вершины стека, очистки используемой памяти. Напишем функцию которая работает с двумя стеками и в зависимости от типа узла - вставляет или удаляет узел в данные стеки (void stack_push(stack* st, stack* res, Node* n)). Обозначим функцию, которая строит бинарное дерево по стеку (void build(tree* t, stack* s)). Напишем функцию печати узла, которая в свою очередь будет использовать функцию "декодирования", то есть по типу узла и значению печатать переменную/число/оператор. Тут же добавим функции печати выражения в стандартном виде и печати самого дерева.

Функция "solving(Node* n)", которая выполняет задание, будет рекурсивно проходить по всему дереву и искать
подходящие по условию узлы, затем создавать новый узел, копировать значения предыдущих узлов, тем самым решая поставленную задачу (будут проверяться потомки узла "^" в поисках правого потомка "-").
В файле "main.c" содержится главная фунцкия "int main()", в которой посимвольно считывается арифметическое
выражение и последовательно заполняется стек "result". Далее по этому стеку строится дерево выражений. Программа выводит дерево исходного выражения, преобразованное выражение в инфиксной форме, дерево преобразованного выражения и завершает свою работу.
Создаём Makefile для компиляции всех трёх файлов. Установим компилятор ("CC = gcc"), выберем стандартные опции компилятора ("CFLAGS = -std=c99 -Wall -Werror"). Далее обозначим цель (all: myprogram) и скомпилируем соответствующие файлы. Затем определим цель clean для удаления выходного файла и всех объектных файлов ("rm -f myprogram *.o"). Сохраним файл и запустим компиляцию в терминале с помощью команды make.

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

Допущен к выполнению работы. Подпись преподавателя _____

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

main.c:

```c
#include "mytree.h"
#include <ctype.h>

int main(){
    stack st = new_stack();
    stack result = new_stack();
    Node* n = create_node(OPERATOR, 1, -1);
    char symb;
    printf("Введите выражение:\n");
    scanf("%c", &symb);
    while (symb != '\n'){
        if (isdigit(symb)) {
            if (n->type == NUMBER || n->type == VARIABLE)
                n->val = n->val * 10 + (symb - '0');
            else{
                stack_push(&st, &result, n);
                n = create_node(NUMBER, symb - '0', -1);
            }
        } else if (isalpha(symb)) {
            if (n->type == NUMBER || n->type == VARIABLE){
                push (&result, n);
                stack_push(&st, &result, create_node(SIGN, 2, 2));
                n = create_node(VARIABLE, -1, -1);
                n->ch = symb;
            }
            else {
                stack_push(&st, &result, n);
                n = create_node(VARIABLE, -1, -1);
                n->ch = symb;
            }
        } else {
            switch(symb) {
                case '(':
                    if (n->type == NUMBER || n->type == VARIABLE)
                        push(&result, n);
                    else
                        stack_push(&st, &result, n);
                    n = create_node(OPERATOR, 1, -1);
                    break;
                case ')':
                    if (n->type == NUMBER || n->type == VARIABLE)
                        push(&result, n);
                    else
                        stack_push(&st, &result, n);
                    n = create_node(OPERATOR, -1, -1);
                    break;
                case '-':
                    if (n->type == NUMBER || n->type == VARIABLE) {
                        push(&result, n);
                        n = create_node(SIGN, 0, 1);
                    } else if (n->type == OPERATOR && n->val != 1) {
                        stack_push(&st, &result, n);
                        n = create_node(SIGN, 0, 1);
                    } else {
                        stack_push(&st, &result, n);
                        push(&result, create_node(NUMBER, -1, -1));
                        n = create_node (SIGN, 2, 2);
                    }
                    break;
                case '+':
                    if (n->type == NUMBER || n->type == VARIABLE)
                        push(&result, n);
                    else
                        stack_push(&st, &result, n);
                    n = create_node(SIGN, 1, 1);
                    break;
                case '*':
                    if (n->type == NUMBER || n->type == VARIABLE)
                        push(&result, n);
                    else
                        stack_push(&st, &result, n);
                    n = create_node(SIGN, 2, 2);
                    break;
                case '/':
                    if (n->type == NUMBER || n->type == VARIABLE)
                        push(&result, n);
                    else
                        stack_push(&st, &result, n);
                    n = create_node(SIGN, 3, 3);
                    break;
                case '^':
                    if (n->type == NUMBER || n->type == VARIABLE)
                        push(&result, n);
                    else
                        stack_push(&st, &result, n);
                    n = create_node(SIGN, 4, 4);
                    break;
            }
        }
        scanf("%c", &symb);
    }
    if (n->type == NUMBER || n->type == VARIABLE)
        push(&result, n);
    else
        stack_push(&st, &result, n);
    stack_push(&st, &result, create_node(OPERATOR, -1, -1));
    tree tree_ = new_tree();
    build (&tree_, &result);
    printf("Дерево исходного выражения:\n");
    printf("\n-------------------------------------------------------------------------------\n");
    print_tree(tree_.root, 1);
    printf("\n-------------------------------------------------------------------------------\n");
    transform(&tree_);
    printf("Преобразованное выражение:\n");
    print_infix(&tree_);
    printf("\n");
    printf("Дерево преобразованного выражения:\n");
    printf("\n-------------------------------------------------------------------------------\n");
    print_tree(tree_.root, 1);
    printf("\n-------------------------------------------------------------------------------\n");
    free_tree(&tree_);
    return 0;
}
```

Makefile:

```
CC = gcc
CFLAGS = -std=c99 -Wall -Werror

all: myprogram
myprogram: main.o mytree.o
	$(CC) $(CFLAGS) -o myprogram main.o mytree.o
main.o: main.c mytree.h
	$(CC) $(CFLAGS) -c main.c
mystack.o: mytree.c mytree.h
	$(CC) $(CFLAGS) -c mytree.c
clean:
	rm -f myprogram *.o
```

mytree.c:

```c
#include "mytree.h"

stack new_stack() {
    stack a;
    a.head = NULL;
    return a;
}

Node* create_node(Type type, int val, int rang) {
    Node* n = (Node*)malloc(sizeof(Node));
    n->type = type;
    n->val = val;
    n->rang = rang;
    n->l = NULL;
    n->r = NULL;
    return n;
}

tree new_tree() {
    tree a;
    a.root = NULL;
    return a;
}

void push(stack* n, Node* a) {
    if (n->head == NULL){
        n->head = a;
        return;
    }
    a->l = n->head;
    n->head = a;
}

Node* delet(stack* n) { //удаление и взятие вершины стека
    Node* t = n->head;
    if (n->head == NULL)
        return NULL;
    if (n->head->l == NULL) {
        n->head = NULL;
        t->l = NULL;
        t->r = NULL;
        return t;
    }
    n->head = t->l;
    t->l = NULL;
    t->r = NULL;
    return t;
}

void stack_push(stack* st, stack* res, Node* n) {
    if (n->type == OPERATOR){
        if (n->val == 1) {
            push (st, n);
        } else {
            Node * t = delet(st);
            while (t->type != OPERATOR || t->val != 1) {
                push (res, t);
                t = delet(st);
            }
            free(t);
            free(n);
        }
    }
    else if (n->type == SIGN) {
        Node * t = delet(st);
        while (t != NULL && n->rang <= t->rang) {
            push(res, t);
            t = delet(st);
        }
            push(st, t);
        push(st, n);
    } else {
        push(res, n);
    }
}

Node* add(Node* n, stack* s) {
    if (n->type == NUMBER || n->type == VARIABLE)
        return n;
    Node * t = n;
    t->l = add(delet(s), s);
    t->r = add(delet(s), s);
    return t;
}

void build(tree* t, stack* s) {//строим дерево по стеку
    t->root = delet(s);
    t->root->l = add(delet(s), s);
    t->root->r = add(delet(s), s);
}

void decoding(Node * n) {
    if (n->type == SIGN) {
        switch(n->val) {
            case 0:
                printf("-");
                break;
            case 1:
                printf("+");
                break;
            case 2:
                printf("*");
                break;
            case 3:
                printf("/");
                break;
            default:
                printf("^");
                break;
        }
    } else if (n->type == VARIABLE) {
        printf ("%c",n->ch);
    } else if (n->type == NUMBER) {
        printf ("%d",n->val);
    }
}

void print_tree(Node* root, int n) {//форматированный вывод дерева
    if (root == NULL) {
        return;
    }
    print_tree(root->r, n + 1);
    for (int i = 0; i < n; i++) printf("\t");
    decoding(root);
    printf("\n");
    print_tree(root->l, n + 1);
}

Node* copy(Node* a){
    Node* n = create_node(a->type, a->val, a->rang);
    n->ch = a->ch;
    if (a->l != NULL)
        n->l = copy(a->l);
    if (a->r != NULL)
        n->r = copy(a->r);
    return n;

}

bool solving(Node* n) {
    if (n->type == SIGN && n->val == 4 && n->l->type == SIGN && n->l->val == 0)
        Node* a = n->r;
        Node* b = n->l->l;
        Node* c = n->l->r;
        Node* a2 = copy(a);
        n->val = 3;
        n->rang = 2;
        n->l->val = 4;
        n->l->rang = 4;
        n->r = create_node(SIGN, 4, 4);
        n->r->r = a2;
        n->r->l = c;
        n->l->r = a;
        n->l->l = b;
        return 1;
    }
    bool p = false;
    bool q = false;
    if (n->l != NULL)
        p = solving(n->l);
    if (n->r != NULL)
        q = solving(n->r);
    if (p || q)
        return 1;
    else
        return 0;
}

void transform(tree* t) {//преобразование выражения
    while (t->root != NULL && solving(t->root));
}


void print_node(Node* n, int l, int r) { //печать узла
    if (n->type == NUMBER || n->type == VARIABLE){
        for (int i = 0; i < l; i++)
            printf("(");
        if (n->type == NUMBER)
            printf("%d", n->val);
        else
            printf("%c", n->ch);
        for (int i = 0; i < r; i++)
            printf(")");
        return;
    }
    if (n->r->type == SIGN && n->r->rang < n->rang)
        print_node(n->r, l + 1, 1);
    else if (n->r->type == 1 || n->r->type == VARIABLE)
        print_node(n->r, l, 0);
    else
        print_node(n->r, l, 0);
    decoding(n);
    if (n->l->type == SIGN && n->l->rang < n->rang)
        print_node(n->l, 1, r + 1);
    else if (n->l->type == 1 || n->l->type == VARIABLE)
        print_node(n->l, 0, r);
    else
        print_node(n->l, 0, r);
}

void print_infix(tree* t) { //печать выражения в обычном виде
    print_node(t->root, 0, 0);
    printf("\n");
}

void free_node(Node* n) {
    if (n->l != NULL)
        free_node(n->l);
    if (n->r != NULL)
        free_node(n->r);
    free(n);
}

void free_tree(tree* tr) {
    free_node(tr->root);
    tr->root = NULL;
}
```

mytree.h:

```c
#ifndef MYTREE
#define MYTREE
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef enum _type {
    OPERATOR,
    NUMBER,
    SIGN,
    VARIABLE
}Type;

typedef struct _node {
    Type type;
    int val, rang;
    /*val - значение узла - число или код для знака;
    rang - приоритет операции;*/
    char ch;
    struct _node *l, *r;//указатели на правый и левый дочерние узлы
}Node;

typedef struct _stack{
    Node* head;
}stack;

typedef struct _tree{
    Node* root;
}tree;

tree new_tree();
stack new_stack();
Node* create_node(Type type, int val, int rang);
void push(stack* now, Node* a);
Node* delet(stack* n);
void stack_push(stack* st, stack* res, Node* now);
Node* add (Node* now, stack* s);
void build(tree* t, stack* s);
void decoding(Node * n);
void print_tree(Node* root, int n);
void print_node(Node* n, int l, int r);
Node* copy (Node* a);
bool solving(Node* n);
void transform(tree* t);
void print_infix(tree* t);
void free_node(Node* n);
void free_tree(tree* tr);
#endif
```

Тестирование:

```
diss@diss-HP-Pavilion-Laptop-15-eh1xxx:~/24$ ./myprogram
Введите выражение:
12 + a - c ^ (15 + e ) - 24 / 3
Дерево исходного выражения:

--------------------------------------------------------------------------------
                        12
                +
                        a
        -
                        c
                ^
                                15
                        +
                                e
    -
                24
            /
                3

--------------------------------------------------------------------------------
Преобразованное выражение:
12+a-c^(15+e)-24/3

Дерево преобразованного выражения:

--------------------------------------------------------------------------------
                        12
                +
                        a
        -
                        c
                ^
                                15
                        +
                                e
    -
                24
            /
                3

--------------------------------------------------------------------------------
```

```
diss@diss-HP-Pavilion-Laptop-15-eh1xxx:~/24$ ./myprogram
Введите выражение:
34 / s * p ^ (15 - 20 * e) + 32 / 45 * c
Дерево исходного выражения:

--------------------------------------------------------------------------------
                        34
                /
                        s
            *
                        p
                ^
                                15
                        -
                                        20
                                *
                                        e
    +
                32
            /
                45
        *
                c

--------------------------------------------------------------------------------
Преобразованное выражение:
34/s*p^15/p^(20*e)+32/45*c

Дерево преобразованного выражения:

--------------------------------------------------------------------------------
                        34
                /
                        s
            *
                                p
                        ^
                                15
                /
                                p
                        ^
                                        20
                                *
                                        e
    +
                32
            /
                45
        *
                c

--------------------------------------------------------------------------------
```

```
diss@diss-HP-Pavilion-Laptop-15-eh1xxx:~/24$ ./myprogram
Введите выражение:
567 / (23 + e) * k ^ (15 - s ^ (24 - 12)) * 32 ^ (t - t ^ 2)
Дерево исходного выражения:

--------------------------------------------------------------------------------
                        567
                /
                                23
                        +
                                e
            *
                        k
                ^
                                15
                        -
                                        s
                                ^
                                                24
                                        -
                                                12
    *
                32
            ^
                        t
                -
                                t
                        ^
                                2

--------------------------------------------------------------------------------
Преобразованное выражение:
567/(23+e)*k^15/k^(s^24/s^12)*32^t/32^t^2

Дерево преобразованного выражения:

--------------------------------------------------------------------------------
                        567
                /
                                23
                        +
                                e
            *
                                k
                        ^
                                15
                /
                                k
                        ^
                                        s
                                ^
                                                24
                                /
                                        s
                                ^
                                                12
    *
                32
            ^
                        t
        /
                32
            ^
                                t
                        ^
                                2

--------------------------------------------------------------------------------
```

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. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10. **Замечания автора** по существу работы _____

11. **Выводы**

В ходе лабораторной работы я углубился в изучение важной части мира программирования - динамических структур. Я научился записывать арифметические выражения в другой записи - обратной польской нотации - с помощью стека. Я ещё раз поработал с бинарными деревьями и научился строить дерево выражений. Научился работать с арифметическим выражением посредством работы с деревом. Как я понял деревья выражений используют для автоматизации рутинных действий. Эти навыки и знания мне определённо понадобятся при дальнейшем изучении программирования.

Недочёты при выполнении задания могут быть устранены следующим образом:_____

Подпись студента_____