Отчёт по лабораторной работе № 24 по курсу 1

студента группы М80-108Б-19 Хренниковой Ангелины , № по списку 23

Адреса www, e-mail, jabber, skype: lina.khrennikova@mail.ru

Работа выполнена: "8" апреля 2020г.

Преподаватель: Поповкин А. В. каф.806

Входной контроль знаний с оценкой

Отчёт сдан "9" апреля 2020 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 №23*): Упростить дробь, сократив в числителе и знаменателе общие переменный и константы: (a*b*3)/(3*b*c) -> a/c; a/(a*b) -> 1/b.

4. **Оборудование** (*лабораторное*):
ЭВМ PC, процессор Intel® Core™ i7-3770 CPU @ 3.40GHz * 8, имя узла сети alise18 с ОП 15974,4 МБ, НМД 345,5 ГБ.
Терминал Gnome адрес 192.168.2.118/24 . Принтер
Другие устройства

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel® Core™ i3-7020U CPU @ 2.30GHz * 4, ОП 8192 МБ, НМД 256 ГБ. Монитор LCD
Другие устройства

5. **Программное обеспечение** (*лабораторное*):
Операционная система семейства UNIX, наименование Ubuntu версия 18.04
Интерпретатор команд Bash версия 4.4.20(1)
Система программирования версия
Редактор текстов Nano версия 2.9.3
Утилиты операционной системы touch, make, cat, ls
Прикладные системы и программы
Местонахождения и имена файлов программ и данных home/stud/olen

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства UNIX, наименование Ubuntu версия 18.04
Интерпретатор команд Bash версия 4.4.19(1)
Система программирования версия
Редактор текстов Emacs версия 25.2.2
Утилиты операционной системы touch, cat, ls, make
Прикладные системы и программы
Местонахождения и имена файлов программ и данных home/lina_tucha/dir/lab_24

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

Создаем структуру для стека(сам элемент, размер стека и вместимость) и функции для создания и удаления стека, добавление и извлечение элемента, структуру для дерева(узел, указатель на правое поддерево, указатель на левое поддерево), структуры для разных типов символов.
В основной части программы определяем приоритеты для операций, распределяем символы по типам и в зависимости от них выстраиваем дерево выражений., печатаем его, преобразуем выражение(удаляем первые совпавшие цифры/буквы из левого и правого поддеревьев) и выводим. При любом отклонении выводим ошибку.

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

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

```
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ make -f makefile
g++ main2.c stack.c stack.h tree.h symbol.h -o 123
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ ./123
(a*b*3)/(3*b*c)

--------------------
/
    *
        *
            a
            b
        3.00
    *
        *
            3.00
            b
        c

(a)/(c)

----------------------
a/(a*b)

--------------------
/
    a
    *
        a
        b

1/(b)

----------------------
(a*b*c)/(a*b*d)

--------------------
/
    *
        *
            a
            b
        c
    *
        *
            a
            b
        d
```

(c)/(d)

(a*f*h*4)/(a*4*h*k)

/
    *
        *
            *
                a
                f
            h
        4.00
    *
        *
            *
                a
                4.00
            h
        k

(f)/(k)

(7*4*2)/(f*2*t)

/
    *
        *
            7.00
            4.00
        2.00
    *
        *
            f
            2.00
        t

(7*4)/(f*t)

(a*d*h)/(w*t*k)

/
    *
        *
            a
            d
        h
    *
        *
            w
            t
        k

(a*d*h)/(w*t*k)

(q*4*u*o)/(q*7*g)

/

```
    *
        *
            *
                q
                4.00
            u
        o
    *
        *
            q
            7.00
        g
```

(4*u*o)/(7*g)

Допущен к выполнению работы. Подпись преподавателя _____

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

```
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ cat makefile
all:
	g++ main2.c stack.c stack.h tree.h symbol.h -o 123

lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ cat main2.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "symbol.h"
#include "stack.h"
#include "tree.h"

// освобождение памяти, удаление стеков
void destructor(STACK *a, STACK *b) {
	stack_delete(a);
	stack_delete(b);
}

// приоретет операндов
int op_priority(char op) {
   switch(op) {
      case OP_MINUS:
      case OP_PLUS:
         return 1;
      case OP_MULT:
      case OP_DIVIDE:
         return 2;
      case OP_POW:
         return 3;
      case OP_UNARY_MINUS:
         return 4;
   }
   return -1;
}

// ассоциативность операндов
OP_ASSOC op_assoc(OP op) {
   switch(op) {
      case OP_MINUS:
      case OP_PLUS:
      case OP_MULT:
      case OP_DIVIDE:
         return ASSOC_LEFT;
      case OP_UNARY_MINUS:
      case OP_POW:
         return ASSOC_RIGHT;
   }
   //return -1;
```

```
}

// перечислимый OP в char; унарный минус учтен!
char op_to_char(OP op) {
   switch(op) {
      case OP_MINUS:
      case OP_PLUS:
      case OP_MULT:
      case OP_DIVIDE:
      case OP_POW:
         return op;
      case OP_UNARY_MINUS:
         return '-';
   }
   return -1;
}

// конвертируем тип int в op
OP int_to_op(int i) {
      switch(i){
            case '+': return OP_PLUS;
            case '*': return OP_MULT;
            case '/': return OP_DIVIDE;
            case '^': return OP_POW;
      }
}

// пробелы
bool is_space(int c) {
   return (c == ' ') || (c == '\t');
}

// пропускает пробелы и читает следующий значащую литеру
int next_char() {
   int c;
   while(is_space(c = getchar())) {} // пустой цикл: все нужное - прямо в условии
   return c;
}

int i=0, g=0;
char a[100];

// обработка следующей литеры
bool next_symbol(symbol *out) {
   // здесь будем хранить тип последней просчитанной литеры
   // надо это затем, чтобы, столкнувшись с унарным минусом, не спутать его с бинарным
   static symb_TYPE prev_type = symb_NONE;

   int c = next_char();

   a[i]=c;
   i++;

   if(c == EOF) {
      out->type = symb_EOF; // на всякий случай
      prev_type = symb_NONE; // тоже на всякий
      return false; // конец ввода
      } else if(c == '\n'){
            out->type = symb_ENDL; // передаем конец строки
      prev_type = symb_NONE;
      return false; // конец ввода
   } else if(c == '.' || (c >= '0' && c <= '9')) {
      // число может начинаться с точки: .9 == 0.9
      ungetc(c, stdin);
      out->type = symb_NUMBER;
      scanf("%f", &(out->data.number)); // читаем значение в структуру
   } else if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
      ungetc(c, stdin);
      out->type = symb_VAR;
      scanf("%[^ \n\t+-*/^()]", out->data.var);
   } else if(c == '(') {
      out->type = symb_LEFT_BR;
```

```
    } else if(c == ')') {
        out->type = symb_RIGHT_BR;
    } else if(c == '+' || c == '*' || c == '/' || c == '^') {
        out->type = symb_OP;
        out->data.op = int_to_op(c);
    } else if(c == '-') {
        out->type = symb_OP;
        // тут надо проверить, унарный ли минус!
        if(prev_type == symb_OP || prev_type == symb_NONE) {
            out->data.op = OP_UNARY_MINUS;
        } else {
            out->data.op = OP_MINUS;
        }
    } else {
        // явная ошибка: не должно быть каких-либо недопустимых символов
        out->type = symb_NONE;
        out->data.c = c;
    }

    // запоминаем в статической переменной тип только что считанного символа
    prev_type = out->type;

    return true;
}

// построение дерева выражения
bool build_tree(TN **tree, STACK *rev) {
    if(stack_empty(rev)) {
        return false;
    }
    symbol t = stack_pop(rev);
    (*tree) = (TN*)malloc(sizeof(TN));
    (*tree)->t = t;

    bool res = true;
    if(t.type == symb_OP) {
        if(t.data.op == OP_UNARY_MINUS) {
            (*tree)->l = NULL;
            res = res && build_tree(&((*tree)->r), rev);
        } else {
            res = res && build_tree(&((*tree)->r), rev);
            // если res или false, то build_tree уже не вызывается
            res = res && build_tree(&((*tree)->l), rev);
        }
    }
    return res;
}

// распечатка дерева выражения
void print_tree(TN *tree, int lev) {
    for(int i = 0; i < lev; i++) {
        printf("\t");
    }
    switch(tree->t.type) {
        case symb_NUMBER:
            printf("%.2lf\n", tree->t.data.number);
            break;
        case symb_VAR:
            printf("%s\n", tree->t.data.var);
            break;
        case symb_OP:
            if(tree->t.data.op == OP_UNARY_MINUS) {
                printf("-\n");
                print_tree(tree->r, lev+1);
            } else {
                printf("%c\n", op_to_char(tree->t.data.op));
                print_tree(tree->l, lev+1);
                print_tree(tree->r, lev+1);
            }
            break;
        default:
            fprintf(stderr, "В дереве таких символов быть уже не должно\n");
```

```
            return;
        }
    }

    // распечатка дерева в форме выражения
    void print_expr(TN *tree) {
        switch(tree->t.type) {
            case symb_NUMBER:
                printf("%.2lf", tree->t.data.number);
                break;
            case symb_VAR:
                printf("%s", tree->t.data.var);
                break;
            case symb_OP:
                if(tree->t.data.op == OP_UNARY_MINUS) {
                    printf("-");
                    print_expr(tree->r);
                } else {
                    printf("(");
                    print_expr(tree->l);
                    printf("%c", op_to_char(tree->t.data.op));
                    print_expr(tree->r);
                    printf(")");
                }
                break;
            default:
                fprintf(stderr, "В дереве таких символов уже быть не должно\n");
                return;
        }
    }

    // функция ввода и обработки дерева выражений
    int enter_and_build_tree() {
        STACK *s, *rev;
        symbol t;
            symb_TYPE checker;

        s = stack_create();
        rev = stack_create();

        while(next_symbol(&t)) {
            switch(t.type) {
                case symb_NONE:
                    fprintf(stderr, "Ошибка: токен %c не  распознан\n", t.data.c);
                            destructor(s, rev);
                    return 1;

                case symb_OP:
                    for(;;) {
                        if(stack_empty(s)) break;
                        symbol top = stack_peek(s);
                        if(top.type != symb_OP) break;

                        if((op_assoc(t.data.op) == ASSOC_LEFT && op_priority(t.data.op) <= op_priority(top.data.op))
                        || (op_assoc(t.data.op) == ASSOC_RIGHT && op_priority(t.data.op) < op_priority(top.data.op))
                        ) {
                            // перекладываем
                            stack_pop(s);
                            stack_push(rev, top);
                        } else {
                            break;
                        }
                    }

                    // кладем новый оператор на верхушку стека
                    stack_push(s, t);
                    break;

                case symb_NUMBER:
                case symb_VAR:
                    // эти вещи сразу пихаем в выходной стек
                    stack_push(rev, t);
```

```
            break;

        case symb_LEFT_BR:
            stack_push(s, t);
            break;

        case symb_RIGHT_BR:
            for(;;) {
                if(stack_empty(s)) {
                    fprintf(stderr, "Ошибка: непарная закрывающая скобка\n");
                                destructor(s, rev);
                    return 2;
                }
                symbol top = stack_peek(s);
                if(top.type == symb_LEFT_BR) {
                    stack_pop(s);
                    break;
                } else {
                    // перекладываем
                    stack_pop(s);
                    stack_push(rev, top);
                }
            }
            break;
    }
}

    checker = t.type; // проверка типа последнего символа

    //обработка конца файла
    //если последний символ - EOF, то стеки очищаются
    if(checker == symb_EOF) {
            destructor(s, rev);
            return 0;
    }
    printf("\n--------------------\n");

while(!stack_empty(s)) {
    t = stack_pop(s);
    if(t.type == symb_LEFT_BR) {
        fprintf(stderr, "Ошибка: непарная открывающая скобка\n");
                destructor(s, rev);
        return 2;
    }
    stack_push(rev, t);
}

// Build tree
if(stack_empty(rev)) {
    fprintf(stderr, "Ошибка: выражение пусто\n");
            destructor(s, rev);
    return 3;
}

TN *root = NULL;
if(!build_tree(&root, rev)) {
    fprintf(stderr, "Ошибка при построении дерева: не найден один из операндов\n");
            destructor(s, rev);
    return 4;
}
if(!stack_empty(rev)) {
    fprintf(stderr, "Ошибка при построении дерева: лишние опернды или операции\n");
            destructor(s, rev);
    return 4;
}

print_tree(root, 0);

printf("\n");
for (int j=0; j<i; ++j) {
    if (a[j]=='/') g=j;
}
```

```
    for (int j=1; j<g-1; ++j) {
        for (int h=g+1; h<i-1; ++h) {
            if ((a[j]>='0'&&a[j]<='9')||(a[j]>='a'&&a[j]<='z')||(a[j]>='A'&&a[j]<='Z')) {
                if (a[j]==a[h]) {
                    a[j]='?'; a[h]='?';
                    if (j!=g-2) {
                        a[j]='?'; a[j+1]='?';
                    }
                    else {
                        a[j-1]='?'; a[j]='?';
                    }
                    if (h!=i-2) {
                        a[h]='?'; a[h+1]='?';
                    }
                    else {
                        a[h-1]='?'; a[h]='?';
                    }
                }
            }
        }
    }
    for (int j=0; j<i; ++j) {
        if (a[j]=='?') continue;
        else printf("%c", a[j]);
    }
    i=0;
    printf("\n----------------------\n");

        destructor(s, rev);

        if(checker == symb_ENDL) return 5;
        else return 6;
}

// программа преобразования выражений в деревья
int main(int argc, char* argv[]) {
        int error_code;

        // ввод и преобразование очередного дерева выражения
        do{
                error_code = enter_and_build_tree();
        }while(error_code);

        printf("\n--------------------\n");

        return 0;
}
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ cat tree.h
#ifndef __tree_h__
#define __tree_h__

#include <stdbool.h>

#include "symbol.h"

// дерево выражений
typedef struct _TN {
    symbol t;           // значение узла
    struct _TN* l;      // указатель на левое поддерево
    struct _TN* r;      // указатель на правое поддерево
} TN;

#endif
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ cat symbol.h
#ifndef __symbol_h__
#define __symbol_h__

// перечислимый тип ассоциативности операции
typedef enum _OP_ASSOC {
    ASSOC_LEFT, ASSOC_RIGHT
} OP_ASSOC;
```

```
// максимальная длина имени переменной
#define VARNAME_LEN 10

// перечислимый тип категории символов(лексем, токенов)
typedef enum _symb_TYPE {
   symb_NONE,    // не вимвол вовсе
      symb_ENDL,        // конец строки
      symb_EOF,         // конец файла
   symb_NUMBER,  // число
   symb_VAR,     // переменная
   symb_OP,      // оператор
   symb_LEFT_BR, // открывающая скобка
   symb_RIGHT_BR // закрывающая скобка
} symb_TYPE;

// перечислимый тип допустимых операций
typedef enum _OP {
   OP_MINUS = '-',
   OP_PLUS = '+',
   OP_MULT = '*',
   OP_DIVIDE = '/',
   OP_POW = '^',
   OP_UNARY_MINUS = '!' // тут что угодно может быть, лишь бы с простым минусом не путать
} OP;

// тип и значение символа
typedef struct {
   symb_TYPE type;                  // тип символа
   union {
      float number;          // если символ - число
      char var[VARNAME_LEN]; // если символ - переменная
      OP op;                 // если символ - опералор
      char c;                // на случай ошибки, здесь будет считанный непонятный char
   } data;
} symbol;

#endif
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ cat stack.h
#ifndef __stack_h__
#define __stack_h__

#include <stdbool.h>

#include "symbol.h"

// стек на массиве
typedef struct {
   symbol *body; // указатель на резидентный массив(вектор)
   int size; // текущий размер стека
   int cap; // максимальная вместимость
} STACK;

STACK *stack_create();
void stack_delete(STACK *stack);
bool stack_empty(STACK *stack);
void stack_push(STACK *stack, symbol t);
symbol stack_pop(STACK *stack);
symbol stack_peek(STACK *stack);

#endif
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab_24$ cat stack.c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "symbol.h"
#include "stack.h"

#define MINSIZE 4

// создание стека
```

```
STACK *stack_create() {
   STACK *stack = (STACK*)malloc(sizeof(STACK));
   stack->cap = MINSIZE;
   stack->size = 0;
   stack->body = (symbol*)malloc(sizeof(symbol) * stack->cap);
   return stack;
}

// удаление стека
void stack_delete(STACK *stack) {
   free(stack->body);
   free(stack);
}

// стек пуст?
bool stack_empty(STACK *stack) {
   return stack->size == 0;
}

// положить на стек
void stack_push(STACK *stack, symbol t) {
   if(stack->size <= stack->cap) {
      stack->cap *= 2;
      stack->body = (symbol*)realloc(stack->body, sizeof(symbol) * stack->cap);
   }

   stack->body[stack->size] = t;
   stack->size++;
}

// снять со стека
symbol stack_pop(STACK *stack) {
   symbol res = stack->body[stack->size - 1];
   stack->size--;

   if(stack->size * 2 < stack->cap && stack->cap > MINSIZE) {
      stack->cap /= 2;
      stack->body = (symbol*)realloc(stack->body, sizeof(symbol) * stack->cap);
   }

   return res;
}

// просмотр верхушки
symbol stack_peek(STACK *stack) {
   return stack->body[stack->size - 1];
}
```

9. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) 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. Замечание автора по существу работы _____

11. Выводы : Я составить программу упрощения дроби, сокращая в числителе и знаменателе общие переменные и константы, с применением деревьев

Недочеты, допущенные при выполнении задания, могут быть устранены следующим образом _____

Подпись студента Хренникова А. С.