Отчет по лабораторной работе № 24
по курсу Практикум программирования

Студент группы М8О-111Б-23 Бугренков Владимир Петрович,
№ по списку 4, Контакты e-mail: vladimir.bugrenkov@yandex.ru
Работа выполнена: «18» мая 2023 г.
Преподаватель: доцент каф. 806 Никулин Сергей Петрович
Отчет сдан « » _____20 ___ г., итоговая оценка _____
Подпись преподавателя _____

1. **Тема:** Деревья выражений
2. **Цель работы:** составить программу выполнения заданных преобразований арифметических выражений с применением деревьев
3. **Задание № 24:**

24. Перемножить дроби:

$$(a/b) * (c/d) \rightarrow (a * c)/(b * d).$$

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_ с ОП __16_ Гб НМД ___512 Гб. Монитор __1920x1080~60Hz
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:

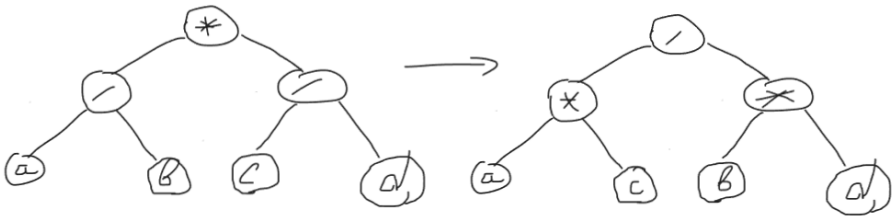
Операционная система семейства _Linux_, наименование _Ubuntu_ версия 22.04.2
интерпретатор команд _GNU bash_ версия 5.1.16.
Система программирования C.
Редактор текстов emacs версия 29.1
Утилиты операционной системы
_____

Прикладные системы и программы Emacs
Местонахождение и имена файлов программ и данных на домашнем компьютере /home/

6. **Идея, метод, алгоритм** решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица]

Составить программу на языке Си, выполняющую четыре действия: ввод выражения, построение дерева, построение выражения из стэка и функцция по заданию

$$(a/b) * (c/d) \rightarrow (a * c)/(b * d).$$



Понадобятся функции:

- main: Главная функция программы, обрабатывает ввод пользователя и управляет программой.
- print_menu: Выводит меню программы.
- print_tree: Выводит дерево выражений в консоль.
- print_expr: Выводит выражение в консоль.
- build_tree: Строит дерево выражений из стека.
- next_char: Считывает следующий непробельный символ из ввода.
- next_symbol: Считывает следующий символ и определяет его тип.
- clearInputBuffer: Очищает буфер ввода.
- simplify_expression2: Упрощает выражение в дереве.
- stack_create: Создает новый стек с начальной емкостью.
- stack_delete: Освобождает память, выделенную под стек.
- stack_empty: Проверяет, пуст ли стек.
- stack_push: Добавляет элемент в стек.
- stack_pop: Удаляет и возвращает последний элемент стека.
- stack_peek: Возвращает последний элемент стека без его удаления.

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

main.c:

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

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

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;
}



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;
}

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;
}

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

int next_char() {
    int c;
    while (is_space(c = getchar())) {
    }
    return c;
}

bool next_symbol(symbol *out) {

    static symb_TYPE prev_type = symb_NONE;

    int c = next_char();
    // сменил EOF на вопросик
    if (c == '?') {
        out->type = symb_NONE;
        prev_type = symb_NONE;
```

```
        return false;
    } else if (c == '.' || (c >= '0' && c <= '9')) {
        // для работы с float: .9 == 0.9
        ungetc(c, stdin);
        out->type = symb_NUMBER;
        scanf("%f", &(out->data.number));
    } else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
        ungetc(c, stdin);
        out->type = symb_VAR;
        // scanf("%[^\n\t+-*/^()]", out->data.var);
        // не английские буквы - нафиг
        scanf("%[a-zA-Z]", out->data.var); // добавляем буквы в переменную
    } 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 = c;
    } else if (c == '-') {
        out->type = symb_OP;
        if (prev_type == symb_OP || prev_type == symb_NONE) {
            out->data.op = OP_UNARY_MINUS;
        } else {
            out->data.op = OP_MINUS;
        }
    } else {
        out->type = symb_NONE;
        out->data.c = c;
    }

    prev_type = out->type;

    return true;
}

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;
            // TODO: исправлял! может ломаться
//          res = res && build_tree(&((*tree)->r), rev);
            res = build_tree(&((*tree)->r), rev);
        } else {
//          res = res && build_tree(&((*tree)->r), rev);
            res = build_tree(&((*tree)->r), rev);
            res = res && build_tree(&((*tree)->l), rev);
        }
    }

    return res;
}

void print_tree(TN *tree, int lev) {

    if (tree->t.type == symb_OP) {
```

```
        print_tree(tree->r, lev + 1);
    }
    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));

            }
            break;
        default:
            fprintf(stderr, "This symlbol must not be in the tree already");
            return;
    }
    if (tree->t.type == symb_OP) {
        print_tree(tree->l, lev + 1);
    }
}

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 {
                if (tree->l->t.type == symb_OP && op_priority(tree->t.data.op) > op_priority(tree->l->t.data.op)) {
                    printf("(");
                    print_expr(tree->l);
                    printf(")");
                } else {
                    print_expr(tree->l);
                }
                //          printf("(");
                //          print_expr(tree->l); 2+3) -> 2+3
                printf("%c", op_to_char(tree->t.data.op));
                //              print_expr(tree->r);// (2+3) -> 2+3
                //              printf(")");
//          if (tree->r->t.type == symb_OP && op_priority(tree->t.data.op) > op_priority(tree->r->t.data.op))
                if (tree->r != NULL && tree->r->t.type == symb_OP &&
                    op_priority(tree->t.data.op) > op_priority(tree->r->t.data.op)) {
                    printf("(");
                    print_expr(tree->r);
                    printf(")");
                } else if (tree->r != NULL) {
                    print_expr(tree->r);
                }
```

```
            }
            break;
        default:
            fprintf(stderr, "This symlbol must not be in the tree already");
            return;
    }
}

void clearInputBuffer() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

void print_menu() {
    puts("-------------------------------------------");
    puts("| 1) Ввести выражение.");
    puts("| 2) Распечатать дерево.");
    puts("| 3) Вывести выражение.");
    puts("| 4) Упростить выражение.");
    puts("| 0) Выход.");
    puts("-------------------------------------------");
    printf("▷▷▷ ");

}


//(a/b)*(c/d)->(a*c)/(b*d)
void simplify_expression2(TN *node) {
    if (node == NULL) {
        return;
    }

    // Рекурсивно обходим левое поддерево
    simplify_expression(node->l);

    // Рекурсивно обходим правое поддерево
    simplify_expression(node->r);

    // Проверка корня
    if (node->t.type == symb_OP && node->t.data.op == OP_MULT) {
        // Проверки
        if (node->l != NULL && node->l->t.type == symb_OP && node->l->t.data.op == OP_DIVIDE &&
            node->r != NULL && node->r->t.type == symb_OP && node->r->t.data.op == OP_DIVIDE) {

            // Создаём ноду деления
            TN *newOpNode = malloc(sizeof(TN));
            newOpNode->t.type = symb_OP;
            newOpNode->t.data.op = OP_DIVIDE;

            // Нода числителя
            TN *newNum = malloc(sizeof(TN));
            newNum->t.type = symb_OP;
            newNum->t.data.op = OP_MULT;
            newNum->l = node->l->l; // Числитель левого деления
            newNum->r = node->r->l; // Числитель правого деления

            // Нода знаменателя
            TN *newDen = malloc(sizeof(TN));
            newDen->t.type = symb_OP;
            newDen->t.data.op = OP_MULT;
            newDen->l = node->l->r; // Знаменатель левого деления
            newDen->r = node->r->r; // Знаменатель правого деления

            // Линкуем ноды
```

```
            newOpNode->l = newNum;
            newOpNode->r = newDen;

            // Заменяем текущий узел на новый узел деления
            *node = *newOpNode;
        }
    }
}

/**
 * точка старта
*/
int main(void) {
    STACK *s, *rev;
    symbol t;
    TN *root = NULL;
    print_menu();
    char choise;
    scanf("%c", &choise);
    while (choise != '0') {
        if (choise == EOF) {
            printf("Bye-bye!");
            break;
        }
        switch (choise) {
            case '1':
                printf("Введи выражение: \n");

                // ============== Ввод выражения ==============
                s = stack_create();
                rev = stack_create();

                while (next_symbol(&t)) {
                    switch (t.type) {
                        case symb_NONE:
                            fprintf(stderr, "Error: symbol %c not recognized\n", t.data.c);
                            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))) {

                                    stack_pop(s);
                                    stack_push(rev, top);
                                } else {
                                    break;
                                }
                            }


                            stack_push(s, t);

                            char *command_user = NULL;
                            if (scanf("%20s", command_user) == EOF) {
                                printf("Woops");
```

```
            }
            break;

        case symb_NUMBER:
        case symb_VAR:

            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, "Error: closing bracket hasn't pair");
                    return 2;
                }
                symbol top = stack_peek(s);
                if (top.type == symb_LEFT_BR) {
                    stack_pop(s);
                    break;
                } else {
                    stack_pop(s);
                    stack_push(rev, top);
                }
            }
            break;
    }
  }

  while (!stack_empty(s)) {
    t = stack_pop(s);
    if (t.type == symb_LEFT_BR) {
      fprintf(stderr, "Error: opening bracket hasn't pair");
      return 2;
    }
    stack_push(rev, t);
  }
  // ============== Конец выражения ==============

  printf("ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО! \n");
  break;

case '2':
  // Распечатать дерево
  if (rev == NULL || stack_empty(rev)) {
    fprintf(stderr, "Error: expression is empty");
    return 3;
  }

  // TN *root = NULL;
  if (!build_tree(&root, rev)) {
    fprintf(stderr, "Error while building tree: don't find one of operands");
    return 4;
  }
  if (!stack_empty(rev)) {
    fprintf(stderr, "Error while building tree: extra operands or opetators");
    return 4;
  }
  if (root != NULL) {
    print_tree(root, 0);
  } else {
```

```
            fprintf(stderr, "Дерево не создано");
            return 3;
        }
        break;

    case '3':
        if (root != NULL) {
            print_expr(root);
            printf("\n");
        } else {
            printf("Выражение не создано.\n");
        }
        break;

    case '4':
        if (stack_empty(rev)) {
            fprintf(stderr, "Error: expression is empty");
            return 3;
        }

        // TN *root = NULL;
        if (!build_tree(&root, rev)) {
            fprintf(stderr, "Error while building tree: don't find one of operands");
            return 4;
        }
        if (!stack_empty(rev)) {
            fprintf(stderr, "Error while building tree: extra operands or opetators");
            return 4;
        }
        // Вызов функции упрощения выражения
        simplify_expression2(root);
        if (root != NULL) {
            printf("Упрощенное выражение: \n");
            print_expr(root);
            printf("\n========\n");
            print_tree(root, 0);
            stack_delete(rev);
            stack_delete(s);
            free(root);
            root = NULL;
        } else {
            fprintf(stderr, "Пуста!!!");
            return 3;
        }
        break;

    case '0':
        printf("Выход из программы.\n");
        exit(0);
        break;

    default:
        printf("Неверный выбор. Попробуйте снова.\n");
  }
  choise = getchar();
  clearInputBuffer();
}

printf("Конец работы программы!\n");

// // УБИРАЕМ НЕНУЖНОЕ
stack_delete(rev);
stack_delete(s);
// free(&t);
```

```
    return 0;
}
```

stack.h

```
#ifndef __stack_h__
#define __stack_h__

#include <stdbool.h>

#include "symbol.h"
#include "utils.h"

typedef struct {
    symbol *body;
    int size; // текущий размер
    int cap; // capacity - 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
```

stack.c:

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

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

#define MINSIZE 4

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;
}

STACK *stack_copy(STACK *original) {
    // Создаем новый стек
    STACK *copy = stack_create();

    // Копируем элементы из исходного стека в новый
    for (int i = 0; i < original->size; i++) {
        stack_push(copy, original->body[i]);
    }

    // Возвращаем копию стека
    return copy;
}

void stack_delete(STACK *stack) {
    // free(stack->body);
    // free(stack);
```

```
    FREE_AND_NULL(stack);

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

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++;
}

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;
}

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

Symbol.h:

```
#ifndef __symbol_h__
#define __symbol_h__


typedef enum _OP_ASSOC {
    ASSOC_LEFT, ASSOC_RIGHT
} OP_ASSOC



#define VARNAME_LEN 10

typedef enum _symb_TYPE {
    symb_NONE,    //
    symb_NUMBER,  //
    symb_VAR,     //
    symb_OP,      //
    symb_LEFT_BR, //
    symb_RIGHT_BR //
} symb_TYPE;

typedef enum _OP {
    OP_MINUS = '-',
    OP_PLUS = '+',
    OP_MULT = '*',
    OP_DIVIDE = '/',
    OP_POW = '^',
```

```
   OP_UNARY_MINUS = '!'
} OP;

typedef struct {
   symb_TYPE type;
   union {
      float number;
      char var[VARNAME_LEN]
      OP op;
      char c;
   } data;
} symbol;

#endif
```

token.h

```
#ifndef __token_h__
#define __token_h__


typedef enum _OP_ASSOC {
   ASSOC_LEFT, ASSOC_RIGHT
} OP_ASSOC;



#define VARNAME_LEN 10

typedef enum _TOK_TYPE {
   TOK_NONE,
   TOK_NUMBER,
   TOK_VAR,
   TOK_OP,        TOK_LEFT_BR,
   TOK_RIGHT_BR
} TOK_TYPE;

typedef enum _OP {
   OP_MINUS = '-',
   OP_PLUS = '+',
   OP_MULT = '*',
   OP_DIVIDE = '/',
   OP_POW = '^',
   OP_UNARY_MINUS = '!'
} OP;

typedef struct {
   TOK_TYPE type;
   union {
      float number;
      char var[VARNAME_LEN];
      OP op;
      char c;
   } data;
} TOKEN;

#endif
```

Tree.h

```
#ifndef __tree_h__
#define __tree_h__

#include <stdbool.h>

#include "symbol.h"



//typedef struct _SN SN;

//struct _TN {
typedef struct SN {
   symbol t;

   struct SN* r;
   struct SN* l;
} TN;

#endif
```

Utils.h:

```
#define FREE_AND_NULL(p) do { free(p); (p) = NULL; } while (0)
```

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

```
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика$ cd '2 Семестр'
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр$ cd ЛР24
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ gcc main.c
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ ./a.out
-------------------------------------------
▉ 1) Ввести выражение.
▉ 2) Распечатать дерево.
▉ 3) Вывести выражение.
▉ 4) Упростить выражение.
▉ 0) Выход.
-------------------------------------------
▷▷▷ 1
```

```
Введи выражение:
(a/b)*(c/d)
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!

▷▷▷ 2
            d
      /
            c
*
            b
      /
            a

▷▷▷ 3
a/b*c/d

▷▷▷ 1
Введи выражение:
(a/b)*(c/d)
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!

▷▷▷ 4
Упрощенное выражение:
a*c/b*d
========
            d
      *
            b
/
            c
      *
            a

▷▷▷ 0
Конец работы программы!
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ gcc main.c
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ ./a.out
-------------------------------------------
▉ 1) Ввести выражение.
▉ 2) Распечатать дерево.
▉ 3) Вывести выражение.
▉ 4) Упростить выражение.
▉ 0) Выход.
-------------------------------------------
▷▷▷ 1
Введи выражение:
(1/2)*(3/4)
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!
```

▷▷▷2

4.00

/

3.00

*

2.00

/

1.00

▷▷▷1

Введи выражение:

(1/2)*(3/4)

ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!

▷▷▷4

Упрощенное выражение:

1.00*3.00/2.00*4.00

========

4.00

*

2.00

/

3.00

*

1.00

▷▷▷ 1

Введи выражение:

(5/v)*(7/s)

ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!

▷▷▷2

s

/

7.00

*

v

/

5.00

▷▷▷1

Введи выражение:

(5/v)*(7/s)

▷▷▷3

5.00/v*7.00/s

▷▷▷4

Упрощенное выражение:

5.00*7.00/v*s

```
========
            s
        *
            v
/
            7.00
        *
            5.00

▷▷▷ 0
```
Конец работы программы!
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ gcc main.c
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ ./a.out
```
-------------------------------------------
```
1) Ввести выражение.
2) Распечатать дерево.
3) Вывести выражение.
4) Упростить выражение.
0) Выход.
```
-------------------------------------------
▷▷▷ 1
Введи выражение:
((a/b)*(c/d))+((g/h)*(j/k))
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!
▷▷▷2
```
                k
            *
                j
        /
                h
            *
                g
+
                d
            *
                c
        /
                b
            *
                a

▷▷▷ 3
a/b*c/d+g/h*j/k
▷▷▷ 5
```
Неверный выбор. Попробуйте снова.
▷▷▷1
```

Введи выражение:
1
((a/b)*(c/d))+((g/h)*(j/k))
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!
▷▷▷4
Упрощенное выражение:
a*c/b*d+g*j/h*k
========

```
                k
            /
                h
        *
                j
            /
                g
+
                d
            /
                b
        *
                c
            /
                a
```

▷▷▷ 1
Введи выражение:
((1/2)*(3/4))-((5/6)*(7/8))
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!

▷▷▷2

```
                8.00
            /
                7.00
        *
                6.00
            /
                5.00
-
                4.00
            /
                3.00
        *
                2.00
            /
                1.00
```

▷▷▷3
3

```
1.00/2.00*3.00/4.00-5.00/6.00*7.00/8.00
▷▷▷ 1
Введи выражение:
((1/2)*(3/4))-((5/6)*(7/8))
ВЫРАЖЕНИЕ БЫЛО ВВЕДЕНО!
▷▷▷4
Упрощенное выражение:
1.00*3.00/2.00*4.00-5.00*7.00/6.00*1.00
========
                8.00
          /
                6.00
     *
                7.00
          /
                5.00
-
                4.00
          /
                2.00
     *
                3.00
          /
                1.00


▷▷▷ 0
Конец работы программы!
qwental@DESKTOP-NKF1EUK:/mnt/c/Users/Holiday/Desktop/Мои материалы/Информатика/2 Семестр/ЛР24$ exit
```

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

Научился работать с деревом выражений, реализовал алгоритм решения предложенной задачи.

Подпись студента _____