Отчёт по лабораторной работе № 24

по курсу «Языки и методы программирования».

Выполнил студент группы М8О-111Б-23: Воробьев Глеб Янович № по списку 5.

Контакты: koshastet13@gmail.com

Работа выполнена: «20» апреля 2024 г.

Преподаватель: каф. 806 Никулин Сергей Петрович

Входной контроль знаний с оценкой: _____

Отчет сдан «21» апреля 2024 г.

Итоговая оценка: _____

Подпись преподавателя: _____

1. **Тема:**
Динамические структуры данных, обработка деревьев; алгоритмы и структуры данных
2. Цель работы:
Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев.
3. Задание:
Вариант 23: упростить дробь, сократив в числители и знаменатели общие переменные и константы
4. Оборудование:
Процессор AMD Ryzen 5 7640HS.
ОП 16 ГБ.
SSD 512 ГБ.
Монитор 2560x1600~165Hz.
5. Программное обеспечение:
Операционная система семейства Unix.
Наименование Ubuntu версия 22.04.3.
Интерпретатор команд GNU bash версия 6.2.0.
Система программирования -.
Редактор текстов Visual Studio Code.
6. Идея, метод, алгоритм решения задачи:
Алгоритм состоит из нескольких частей:

1. Чтение и парсинг входного выражения:
   - Функция next_symbol читает следующий символ выражения и возвращает его тип и значение.
   - Символы могут быть числами, операторами, переменными, скобками или специальными типами EOF или ENDL.

2. Обработка выражения и построение обратной польской нотации (ОПН):
   - Выражение читается посимвольно, и каждый символ обрабатывается в соответствии с его типом.
   - Для операторов учитывается их приоритет и ассоциативность.
   - Скобки обрабатываются отдельно, ищется соответствие между открывающими и закрывающими скобками.
   - Результатом является два стека: стек операторов и стек для ОПН.

3. Построение дерева выражений:
   - Используя стек с ОПН, строится дерево выражений.
   - Функция build_tree рекурсивно создает узлы дерева.

4. Печать и упрощение дерева выражений:
   - Дерево можно распечатать с помощью функции print_tree.
   - Упрощение дерева делается с помощью функции simplify, которая сокращает одинаковые термины в числителе и знаменателе.

5. Печать упрощенного выражения:

- После упрощения дерева функция print_expr печатает финальное выражение.

6. Очистка памяти:
- В конце работы с деревом выражений память, занимаемая деревом, освобождается.

7. Цикл ввода-вывода:
- Основной алгоритм запускается в цикле, позволяя пользователю вводить выражения до получения сигнала конца файла (EOF).

В коде также присутствуют вспомогательные функции для работы со стеками, узлами дерева и символами.

7. Сценарий выполнения работы:

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 категорий символов (лексем, токенов)
typedef enum _symb_TYPE {
   symb_NONE,     // не символ вовсе (C) Дубинин А.В.
   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 = '!' // а тут что угодно может быть, главное, чтобы с
простым минусом не путать (C) Дубинин А.В.
} 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 - оператор
        char c;                 // на случай ошибок, здесь будет считанный
непонятный char
    } data;
} symbol;

#endif
```

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
```

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];
}
```

tree.h

```h
#ifndef __tree_h__
#define __tree_h__

#include <stdbool.h>

#include "symbol.h"

typedef struct _TN {
   symbol t;
   struct _TN* l;
   struct _TN* r;
} TN;

#endif
```

main.c

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

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

void destructor(STACK *a, STACK *b) {
   stack_delete(a);
   stack_delete(b);
}

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;
}

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

bool is_space(int c) {
   return (c == ' ') || (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();

   if(c == EOF) {
       out->type = symb_EOF;
       prev_type = symb_NONE;
       return false;
   } else if(c == '\n'){
       out->type = symb_ENDL;
       prev_type = symb_NONE;
       return false;
   } else if(c == '.' || (c >= '0' && c <= '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);
   } 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;
       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;
            res = res && build_tree(&((*tree)->r), rev);
        } else {
            res = res && build_tree(&((*tree)->r), rev);
            res = res && build_tree(&((*tree)->l), rev);
        }
    }
    return res;
}

void print_tree(TN *tree, int lev) {
    if (tree == NULL) return;

    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 {
                printf("(");
                print_expr(tree->l);
                printf("%c", op_to_char(tree->t.data.op));
                print_expr(tree->r);
                printf(")");
            }
            break;
        default:
            fprintf(stderr, "This symlbol must not be in the tree
already\n");
            return;
    }
}
int compareSymbols(symbol* s1, symbol* s2) {
    if (s1->type != s2->type) {
        return 0;
    }

    if (s1->type == symb_NUMBER)
        return (s1->data.number == s2->data.number);
    else if (s1->type == symb_VAR)
        return (strcmp(s1->data.var, s2->data.var) == 0);
    else
        return 0;
}
```

```
TN *findParent(TN* root, TN* child) {
    if (root == NULL || root == child) {
        return NULL;
    }

    if (root->l == child || root->r == child) {
        return root;
    }

    TN* leftSearch = findParent(root->l, child);
    if (leftSearch != NULL) {
        return leftSearch;
    }

    return findParent(root->r, child);
}

TN *findTN(TN *root, symbol key)
{
    if (root == NULL)
        return NULL;

    if (compareSymbols(&(root->t), &key))
        return root;

    TN *found = findTN(root->l, key);
    if (found != NULL)
        return found;

    return findTN(root->r, key);
}

void deletNode(TN *parent, TN *node) {
    if (node == NULL) return;

    if (node->l != NULL) {
        deletNode(node, node->l);
    }
    if (node->r != NULL) {
        deletNode(node, node->r);
    }

    if (parent != NULL) {
        if (parent->l == node) {
            parent->l = NULL;
        } else if (parent->r == node) {
            parent->r = NULL;
        }
```

```
    }

    free(node);
}

TN *create_one_node() {
    TN *one = (TN *)malloc(sizeof(TN));
    if (one) {
        one->l = NULL;
        one->r = NULL;
        one->t.type = symb_NUMBER;
        one->t.data.number = 1.0;
    }
    return one;
}


void collapse_mult_nodes(TN **node, TN *root) {
    if (*node == NULL) return;

    collapse_mult_nodes(&((*node)->l), root);
    collapse_mult_nodes(&((*node)->r), root);

    if ((*node)->t.type == symb_OP && (*node)->t.data.op == OP_MULT) {

        if ((*node)->l && (*node)->r == NULL) {
            TN *new_node = (*node)->l;
            free(*node);
            *node = new_node;
        } else if ((*node)->r && (*node)->l == NULL) {
            TN *new_node = (*node)->r;
            free(*node);
            *node = new_node;
        } else if ((*node)->r == NULL && (*node)->l == NULL) {
            TN *p = findParent(root, (*node));
            deletNode(p, *(node));
        }
    }
}

void deletTree(TN *root)
{
    if (root == NULL)
        return;

    deletTree(root->l);
    deletTree(root->r);
```

```
    free(root);
}

void equal_one(TN *root)
{
    if ((root->l && root->l->t.type == symb_NUMBER &&
root->l->t.data.number == 1.00) && (root->r && root->r->t.type ==
symb_NUMBER && root->r->t.data.number == 1.00)) {
        deletTree(root->l);
        deletTree(root->r);

        root->l = NULL;
        root->r = NULL;

        root->t.type = symb_NUMBER;
        root->t.data.number = 1.00;
    }
}

void adding_one(TN *root) {
    if (root == NULL) return;

    if (root->l == NULL && root->r == NULL && root->t.type == symb_OP) {
        root->t.type = symb_NUMBER;
        root->t.data.number = 1.0;
        return;
    }

    if (root->l && root->l->l == NULL && root->l->r == NULL &&
root->l->t.type == symb_OP) {
        free(root->l);
        root->l = create_one_node();
    }
    if (root->r && root->r->l == NULL && root->r->r == NULL &&
root->r->t.type == symb_OP) {
        free(root->r);
        root->r = create_one_node();
    }

    if (root->l == NULL) {
        root->l = create_one_node();
    }
    if (root->r == NULL) {
        root->r = create_one_node();
    }

    equal_one(root);
}
```

```
void collectTerms(TN *root, symbol *terms[], int *termsSize) {
   if (root == NULL) return;

   if (root->t.type == symb_NUMBER || root->t.type == symb_VAR) {
       terms[*termsSize] = &root->t;
       (*termsSize)++;
   } else {
       collectTerms(root->l, terms, termsSize);
       collectTerms(root->r, terms, termsSize);
   }
}


void simplify(TN* root, symbol *numerator[], symbol *denominator[]) {
   int numSize = 0;
   int denSize = 0;
   TN *tmp = root;
   collectTerms(tmp->l, numerator, &numSize);

   collectTerms(tmp->r, denominator, &denSize);

   TN *finded;
   TN *parent;
   for (int i = 0; i < numSize; i++) {
       for (int j = 0; j < denSize; j++) {
           if (numerator[i] != NULL && denominator[j] != NULL &&
compareSymbols(numerator[i], denominator[j])) {

               finded = findTN(root->l, *numerator[i]);
               if (finded != NULL) {
                   parent = findParent(root, finded);
                   deletNode(parent, finded);
               }

               finded = findTN(root->r, *denominator[j]);
               if (finded != NULL) {
                   parent = findParent(root, finded);
                   deletNode(parent, finded);
               }
               numerator[i] = NULL;
               denominator[j] = NULL;
           }
       }
   }

   collapse_mult_nodes(&root, root);
   adding_one(root);
```

```
}

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, "Error: symbol %c not
recognized\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))
                   ) {
                       stack_pop(s);
                       stack_push(rev, top);
                   } else {
                       break;
                   }
               }

               stack_push(s, t);
               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\n");
                        destructor(s, rev);
                        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;
        }
    }

    checker = t.type;

    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, "Error: opening bracket hasn't pair\n");
            destructor(s, rev);
            return 2;
        }
        stack_push(rev, t);
    }

    // Build tree
    if(stack_empty(rev)) {
        fprintf(stderr, "Error: expression is empty\n");
        destructor(s, rev);
        return 3;
    }

    TN *root = NULL;
```

```
    if(!build_tree(&root, rev)) {
        fprintf(stderr, "Error while building tree: don’t find one of
operands\n");
        destructor(s, rev);
        return 4;
    }
    if(!stack_empty(rev)) {
        fprintf(stderr, "Error while building tree: extra operands or
opetators\n");
        destructor(s, rev);
        return 4;
    }

    print_tree(root, 0);
    destructor(s, rev);
    printf("\n-------------------\n");
    symbol *numerator[100];
    symbol *denominator[100];
    simplify(root, numerator, denominator);
    print_tree(root, 0);

printf("\n=================================================================\n"
);
    print_expr(root);
    deletTree(root);
    root = NULL;
    printf("\n\n");
    if(checker == symb_ENDL) return 5;
    else return 6;
}

int main(int argc, char* argv[]) {
    int error_code;

    do{
        error_code = enter_and_build_tree();
    }while(error_code);

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

    return 0;
}
```

Допущен к выполнению работы. **Подпись преподавателя _____**

8. **Распечатка протокола:**
xxxkoshaster@YES-MAN:~/Documents/Zayks/lb 5/20200320ET$ make
gcc -g main.c stack.c stack.h tree.h symbol.h

```
xxxkoshaster@YES-MAN:~/Documents/Zayks/lb 5/20200320ET$ ./a.out
(a*b*3)/(3*b*c)

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

--------------------
      c
/
      a

====================================================================
(a/c)

a/(a*b)

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

--------------------
      b
/
      1.00

====================================================================
(1.00/b)

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

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

--------------------
      d
/
      a

====================================================================
(a/d)

(2*3*x)/(3*x*5)

--------------------
            5.00
      *
                  x
            *
                  3.00
/
            x
```

```
      *
            3.00
         *
            2.00

------------------
      5.00
/
      2.00
```

==========

(2.00/5.00)

(x*x*x)/(x*x*x)

```
------------------
         x
      *
            x
         *
            x
/
         x
      *
            x
         *
            x

------------------
1.00
```

==========

1.00

(a*b*c)/(x*y*z)

```
------------------
         z
      *
            y
         *
            x
/
         c
      *
            b
         *
            a

------------------
         z
      *
            y
         *
            x
/
         c
      *
            b
         *
            a
```

==========

(((a*b)*c)/((x*y)*z))

(x*y*z)/(x*y*z)

```
------------------
         z
      *
            y
         *
            x
/
```

```
            z
      *
                  y
            *
                  x

--------------------
1.00
```

==========

```
1.00
```

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

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

--------------------
      b
/
      a
```

==========

```
(a/b)
```

(2*a*3*b)/(3*a*4*b)

```
--------------------
            b
      *
                  4.00
            *
                        a
                  *
                        3.00
/
            b
      *
                  3.00
            *
                        a
                  *
                        2.00

--------------------
      4.00
/
      2.00
```

==========

```
(2.00/4.00)
```

(1*a*b*1)/(a*b*1)

```
--------------------
            1.00
      *
                  b
            *
                  a
/
            1.00
      *
                  b
```

```
          *
                a
            *
                1.00

------------------
1.00

==========================================================================
1.00

1/1

------------------
      1.00
/
      1.00

------------------
1.00

==========================================================================
1.00

------------------
      a
/
      a

------------------
1.00

==========================================================================
1.00
```

9. Дневник отладки

№	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание

10. Замечания автора:
По существу работы: замечания отсутствуют.

11. Выводы:
В ходе выполнения лабораторной работы я научился работать с двоичными деревьями в СП Си и обрабатывать выражения заданным образом.

Подпись студента _____