# Отчет по лабораторной работе № 24 по курсу
# Фундаментальная информатика

Студент группы М8О-104Б-22 Зайцев Данила Евгеньевич, № по списку 8

Контакты www, e-mail, icq, skype zaitsev.danila@yandex.ru

Работа выполнена: « 6 » ноября 2023 г.

Преподаватель: асп. каф. 806 Потенко М.А.

Входной контроль знаний с оценкой _____

Отчет сдан «  »_____202___г., итоговая оценка ___

Подпись преподавателя _____

1. **Тема:** Программирование на языке Си**.**

2. **Цель работы:** Составление и отладка программы на языке Си с вводом арифметического выражения, выполнения с ним манипуляция согласно варианту, демонстрация промежуточных результатов и вывод выражения после маниуляций.

3. **Задание** (*вариант №* 15): Убрать из выражения все сомножители, равные 1.

4. **Оборудование** (*лабораторное*):
   ЭВМ_____, процессор_____, имя узла сети_____с ОП_____Мб,
   НМД_____Мб. Терминал_____адрес_____. Принтер _____
   Другие устройства _____
   _____

   *Оборудование ПЭВМ студента, если использовалось:*
   Процессор Intel(R) Core(TM) i9-9880H с ОП 32 Гб, ssd 938 Гб. Монитор IPS 1920*1080
   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 20.04 LTS, интерпретатор команд bash версия 5.1.16
   Система программирования C версия _____
   Clion _____
   Утилиты операционной системы Терминал _____

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

   Сначала проверим строку на валидность (что это реальное арифметическое выражение), для чего разобьем её на токены , после через алгоритм Дейкстры переведем в обратную польскую нотацию и после в дерево выражения. Далее проходя по дереву ищем листья, где значения листа 1, а родитель «*», заменяем на второго наследника «*», после чего собираем выражение обратно и выводим его.

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

   Подключим нужные для написания программы библиотеки: stdio.h, string.h, ctype.h, stdlib.h, math.h. Также создадим свои собственные библиотеки для удобства работы с алгоритмом Дейкстры. Нам потребуется структура стека на массиве s21_stack_char.h, а также стандартные функции для работы с ней: добавление, изъятие, просмотр верхнего значения, инициализация, очистка, переворачивание; также нам потребуется библиотека для проверки корректности ввода s21_parser.h, содержащая функции проверки строки на корректность, проерка на корректность функции, взятие приоритета оператора, проверка на корректность числа; понадобится библиотека для перевода из вида стека в вид дерева s21_calculation.h; а также потребуется структура дерева на массиве нод s21_stack_node.h, в которой также описана структура дерева и стека нод, а также основные функции: добавления ноды, очистка стека нод, очистка дерева, вывод дерева, перевод в вид выражения из дерева, выполнения преобразования согласно заданию.
   В main.c объявляем два стека: временный и результирующий. Временный нужен для алгоритма Дейкстры, туда будут складываться операторы. Передаем прочитанную строку и два стека в функцию проверки корректности, где попутно переводится выражение в стековый вид. После этого проверяется, что ошибок не было, далее создаем структуру дерева, переворачиваем результирующий стек согласно алогритма и переводим из стекового вида в вид дерева. Выводим дерево. Проводим операцию убирания умножения на 1 и снова выводим дерево. Чистим память

*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

main.c

```c
#include <stdio.h>
#include "s21_calculation.h"
#include "s21_parser.h"

int main() {
    //Выделяем память на вводимую строку
    char *str = calloc( count: 256,  size: sizeof(char));
    //Считываем строку
    scanf("%s", str);
    //Создаем строку состояния
    char *res = calloc( count: 256,  size: sizeof(char));
    res = strcpy(res, "Correct\n");
    //Создаем необходимые структуры:
    //Временный и результирующий стеки
    s21_stack_char TMP, RESULT;
    //Инициализируем их
    s21_init_stack_char( stack: &TMP);
    s21_init_stack_char( stack: &RESULT);
    //Запускаем функцию проверку корректности строки и перевода в стековый вид
    s21_parse( stackChar: &TMP,  stackResult: &RESULT,  string: str);
    //Если ошибок нет
    if (RESULT.is_correct && TMP.is_correct) {
        //Создаем дерево выражений
        s21_node *result;
        //Создаем ещё один временный стек, куда перезапишем развернутый результирующий стек
        s21_stack_char TMP1;
        s21_reverse_stack_char( stack: &RESULT);
        //Скопируем в результирующий стек его развернутую версию
        s21_copy_stack( stack_src: RESULT,  stack_dst: &TMP1);
        //Создадим дерево выражений
        result = s21_calculation( stackChar: &TMP1,  x: NULL);
        //Если создалось корректно
        if (TMP1.is_correct == 1) {
            //Выведем дерево
            s21_show_tree( tree: result,  i: 0);
            //Сделаем удаление всех множителей равных 1
            s21_prime( tree: &result);
            // Покажем изменения
            s21_show_tree( tree: result, i: 0);
            //Переведем выражение в обратное состояние
            char *r = reverse( tree: result);
            //Выведем его
            printf("%s\n", r);
```

```
44              free(r);
45          // При различных ошибках выведем соответсвующее значение
46          } else if (TMP1.is_correct == 1) {
47              res = strcpy(res, "Not correct\n");
48          } else {
49              res = strcpy(res, "Деление на 0\n");
50          }
51      //Чистим память
52          s21_remove_stack_char( stack: &TMP1);
53          if(result != NULL) {
54              s21_remove_tree( tree: &result);
55          }
56      } else {
57          res = strcpy(res, "Not correct\n");
58      }
59      printf("%s", res);
60      free(res);
61      free(str);
62      s21_remove_stack_char( stack: &TMP);
63      s21_remove_stack_char( stack: &RESULT);
64      return 0;
65  }
```

s21_calculation.h

s21_calculation.h

```
1   #include "s21_stack_char.h"
2   #include "s21_stack_node.h"
3   #include <ctype.h>
4   #include <math.h>
5
6   #ifndef S21_CALCULATION_H
7   #define S21_CALCULATION_H
8
9   #define S21_INF 1.0 / 0.0
10
11  s21_node *s21_calculation(s21_stack_char *stackChar, double *x);
12  double s21_get_operator(char *opr, double first, double *second);
13  double s21_get_function(char *fun, double arg);
14  void s21_time_to_clean_up(s21_stack_char *stackChar);
15
16  #endif // S21_CALCULATION_H
17
```

s21_calculation.c

s21_calculation.c

```c
#include "s21_calculation.h"

//Функция очистки стека при некорректности для избежания утечек
void s21_time_to_clean_up(s21_stack_char *stackChar) {
    while (!s21_is_empty_stack_char( stack: *stackChar)) {
        char *tmp = s21_pop_stack_char( stack: stackChar);
        free(tmp);
    }
    s21_ERROR( stack: stackChar);
}

//Функция составления дерева выражений
//s21_stack_char *stackChar - Полученный результирующий стек
// Переменная x, если нужно вместо x вставить значение (в данной реализации не используется
s21_node *s21_calculation(s21_stack_char *stackChar, double *x) {
    //Выводимое дерево
    s21_node *res = NULL;
    //Временный стек
    s21_stack_node TMP;
    //Инициализируем дерево
    s21_init_stack_node( stack: &TMP);
    //Переводим стек в дерево выражений
    while (!s21_is_empty_stack_char( stack: *stackChar)) {
        //Получаем верхнее значение
        char *tmp = s21_pop_stack_char( stack: stackChar);
        //Если число или константа
        if (isdigit( c: tmp[0]) || *tmp == 'e' || *tmp == 'P') {
            //Пушим во временный стек
            s21_push_stack_node( stack: &TMP,  str: tmp);
        // Если функция
        } else if (isalpha( c: tmp[0]) && tmp[0] != 'm') {
            //Если во временном стеке ничего нет, значит ошибка
            if (s21_is_empty_stack_node( stack: TMP)) {
                free(tmp);
                s21_time_to_clean_up(stackChar);
                break;
            } else {
                //Иначе пушим во временный стек
                s21_push_stack_node( stack: &TMP,  str: tmp);
            }
        // Если оператор
        } else {
            //Если во временном стеке ничего нет, значит ошибка
```

s21_calculation.c

```c
            if (s21_is_empty_stack_node( stack: TMP)) {
                free(tmp);
                s21_time_to_clean_up(stackChar);
                break;
            //Если не унарный минус
            } else if (*tmp != '~') {
                // Если во временном стеке меньше 2 чисел, то ошибка
                if (TMP.size < 2) {
                    free(tmp);
                    s21_time_to_clean_up(stackChar);
                    break;
                }
                //Иначе пушим значения
                s21_push_stack_node( stack: &TMP,  str: tmp);
            // Если же унарный минус и стек не пустой, то пушим во временный стек
            } else {
                s21_push_stack_node( stack: &TMP,  str: tmp);
            }
        }
    }
    // Если прошла проверка корректности
    if (stackChar->is_correct == 1) {
        //Переводим стек в вид дерева
        res = s21_pop_stack_node( stack: &TMP);
        //Чистим стек
        s21_remove_stack_node( stack: &TMP);
    }
    //Выводим дерево
    return res;
}
```

s21_parser.h

s21_calculation.c | s21_parser.h

```c
#ifndef S21_PARSER_H
#define S21_PARSER_H

#include "s21_stack_char.h"
#include <ctype.h>
#include <string.h>

void s21_parse(s21_stack_char *stackChar, s21_stack_char *stackResult,
               const char *string);
void s21_push_double(s21_stack_char *stackChar, char *string);
void s21_push_fun(s21_stack_char *stackChar, char *string);
void s21_push_operator(s21_stack_char *stackChar, s21_stack_char *stackResult,
                       char *string, int un_min);
int s21_get_priority(char *opr);

#endif // S21_PARSER_H
```

s21_parser.c

s21_parser.c

```c
#include "s21_parser.h"

// функция проверки на валидность входной строки
// s21_stack_char *stackChar - врменный стек, необходимой для проверки корректности и зане
// s21_stack_char *stackResult - результирующий стек
// const char *string - входная строка
void s21_parse(s21_stack_char *stackChar, s21_stack_char *stackResult,
               const char *string) {
    // переменная для проверки на унарный минус
    // в данном алгоритме унарный минус существует только в двух ситуациях:
    // 1) Начало строки
    // 2) После открывающей скобки
    // ситуации по типу 1*-1 считаются ошибкой, корректная запись: 1*(-1)
  int un_min = 1;
  // вводим "бегунок" по строке" отвечающий за текущее положение в строке
  char *tmp = (char *)string;
  // Цикл, пока не дойдем до конца строки
  for (; *tmp != '\0'; tmp++) {
      // Алгоритм считает эквивалентными записи "1+1" и "1 + 1"
    if (*tmp == ' ') {
      continue;
        //  Алгоритм поддерживает переменные типа
        //  Pi - Число пи
        //  e - Число e
        //  x - Переменная; в данном случаи пропускается и в дальнейшем будет выставлено и
        //  или дополнительно указанное
    } else if (isdigit( c: *tmp) || *tmp == 'P' || *tmp == 'e' || *tmp == 'x') {
      //числа и константы сразу добавляются в результирующий стек согласно алгоритму Дейкс
      s21_push_double( stackChar: stackResult,  string: tmp);
      // Двигаем бегунок на размер числа или длину названия константы
      if (isdigit( c: *tmp)) {
        for (; isdigit( c: *tmp) || *tmp == '.'; tmp++);
      } else {
        for (; isalpha( c: *tmp); tmp++)
          ;
      }
      tmp--;
      //Так как ввели число, по условию далее унарного минуса быть не может
      un_min = 0;
    //Проверка на функцию, типа sin(), tan() и т.д. Общим словом тригонометрические (так к
    } else if (isalpha( c: *tmp) && *tmp != 'm') {
      //Так как у функций есть приоритет, сначала кидаем их во временный стек
      s21_push_fun(stackChar,  string: tmp);
```

s21_parser.c

```c
        //Пропускаем длину названия функции
        for (; isalpha( c: *tmp); tmp++)
          ;
        tmp--;
        //Унарного минуса уже быть не может (Тут после будет отдельно проверка на "(", а зна
        un_min = 0;
      // Остаются только операторы
      } else {
        s21_push_operator(stackChar, stackResult, string: tmp, un_min);
        if (*tmp == '(')
            //Появление унарного минуса при открывающейся скобке
          un_min = 1;
        else
          un_min = 0;
        // Так как у нас есть операция mod, то надо пройти её длину
        if (isalpha( c: *tmp)) {
          for (; isalpha( c: *tmp); tmp++)
            ;
          tmp--;
        }
      }
    }
    //По окончанию пробега по строке происходит проверка на коректнсоть строки, а также полу
    while (!s21_is_empty_stack_char( stack: *stackChar)) {
        // Так как при добавлении в стек ")" мы удаляем оттуда "(", то появление в стеке "("
      if (strcmp(s21_show_top_char( stack: stackChar), "(") == 0)
        s21_ERROR( stack: stackResult);
      //Разворачиваем дополнительный стек согласно алгоритма Дейкстры
      s21_push_stack_char( stack: stackResult, symbols: s21_pop_stack_char( stack: stackChar));
    }
}

//Функция добавления числа в стек
//Внтри функции также проверяется, что число существующего формата: 1.1 или 1, а не 1.1.1
//s21_stack_char *stackChar - стек, куда добавляем число
//char *string - исходная строка
void s21_push_double(s21_stack_char *stackChar, char *string) {
  // Счетчик длины числа
  int i = 0;
  //Местный "бегунок"
  char *tmp = string;
  //Специальное условие для констант
  if (isalpha( c: *tmp)) {
```

s21_parser.c

```c
    for (; isalpha( c: *tmp); tmp++) {
      i++;
    }
  // Условие для чисел
  } else {
    // Переменная для учета точки
    int is_p = 0;
    // Бежим по числу
    for (; isdigit( c: *tmp) || *tmp == '.'; tmp++, i++) {
      // Проверяем, что точка встречается впервые
      if (*tmp == '.' && is_p == 0)
        is_p = 1;
      //Иначе помечаем ошибкой
      else if (*tmp == '.' && is_p == 1) {
        s21_ERROR( stack: stackChar);
      }
    }
  }
  //Выдделяем память на новое число
  char *tmp_number = calloc( count: 256,  size: sizeof(char));
  //копируем его в новую память
  tmp_number = strncpy(tmp_number, string, i);
  //проверяем, что корректный формат
  if (isdigit( c: *tmp_number) || s21_is_fun( fun: tmp_number)) {
    // Пушим в наш стек
    s21_push_stack_char( stack: stackChar,  symbols: tmp_number);
    if (strcmp(tmp_number, "x") == 0)
      // Если же встречаем x, то делаем пометку
      stackChar->is_x++;
  // иначе помечаем ошибкой
  } else {
    s21_ERROR( stack: stackChar);
    free(tmp_number);
  }
}

//s21_stack_char *stackChar - стек, куда добавляем
//char *string - исходная строка
void s21_push_fun(s21_stack_char *stackChar, char *string) {
  // счетчик длины функции
  size_t i = 0;
  // местный "бегунок"
  char *tmp = string;
```

s21_parser.c

```
130     for (; isalpha( c: *tmp); tmp++) {
131       i++;
132     }
133     // копируем найденную функцию
134     char *tmp_fun = calloc( count: 256,  size: sizeof(char));
135     tmp_fun = memcpy(tmp_fun, string, i);
136     // Проверяем, что функция валидная (из пула)
137     if (s21_is_fun( fun: tmp_fun)) {
138       // Если валидная, то пушем
139       s21_push_stack_char( stack: stackChar,  symbols: tmp_fun);
140     } else {
141       // Если не валидная, то помечаем ошибкой
142       s21_ERROR( stack: stackChar);
143     }
144   }
145
146
147   //функция добавления операторов
148   //s21_stack_char *stackChar - временный стек
149   //s21_stack_char *stackResult -результирующйи стек
150   //char *string - исходная строка
151   //int un_min - текущее состояние унарного минуса
152   void s21_push_operator(s21_stack_char *stackChar, s21_stack_char *stackResult,
153                          char *string, int un_min) {
154     //создаем бегунок, которой либо равен 1, если не mod или 0
155     size_t i = isalpha( c: *string) ? 0 : 1;
156     char *tmp_s = string;
157     for (; isalpha( c: *tmp_s); tmp_s++) {
158       i++;
159     }
160     //копируем оператор
161     char *tmp_op = calloc( count: i,  size: sizeof(char));
162     tmp_op = memcpy(tmp_op, string, i);
163     //проверяем приоритетность
164     int priority = s21_get_priority( opr: tmp_op);
165     int tmp;
166     //отдельно проверяем на унарный минус и тогда меняем приоритет и символ, чтоб отличать о
167     if (un_min && *tmp_op == '-') {
168       *tmp_op = '~';
169       priority = 3;
170     }
171     //проверка на невозможные варианты:
172     //неизвестный оператор
```

s21_parser.c

```
173     //оператор начинается на m но сам оператор не mod
174     if (priority == -1 ||
175         (priority == 2 && isalpha( c: *string) && !s21_is_fun( fun: tmp_op))) {
176       free(tmp_op);
177       //Выдаем ошибку
178       s21_ERROR( stack: stackResult);
179     // Если не закрывающаяся скобка или не второй аргумент
180     } else if (priority != 4) {
181       // Согласно алгоритма, мы вытаскиваем из временного стека в результирующий все перемен
182       // Выше добавляемого, за исключением открывающей скобки
183       while ((tmp = s21_get_priority( opr: s21_show_top_char( stack: stackChar))) >= priority &&
184              tmp != -1 && priority != 5 && tmp != 5) {
185         s21_push_stack_char( stack: stackResult,  symbols: s21_pop_stack_char( stack: stackChar));
186       }
187       //Добавляем во временный стек наш оператор
188       s21_push_stack_char( stack: stackChar,  symbols: tmp_op);
189     // Если закрывающая скобка
190     } else {
191       // Вытаскиваем из временного стека все операторы до встречи с открывающей скобкой
192       while ((tmp = s21_get_priority( opr: s21_show_top_char( stack: stackChar))) != 5 &&
193              tmp != 4 && tmp != -1) {
194         s21_push_stack_char( stack: stackResult,  symbols: s21_pop_stack_char( stack: stackChar));
195       }
196       // Если не встретилась открывающая скобка, то выдаем ошибку
197       if (tmp == -1) {
198         s21_ERROR( stack: stackResult);
199       } else {
200         // Если открывающая скобка и при этом наш оператор закрывающая скобка, то удаляем от
201         if (tmp == 5 && *tmp_op == ')')
202           free(s21_pop_stack_char( stack: stackChar));
203         //Проверяем временный стек на пустоту (это нужно, чтоб обозначить, что скобка для ар
204         if (!s21_is_empty_stack_char( stack: *stackChar)) {
205           // Проверяем, что там лежит функция
206           if (isalpha( c: s21_show_top_char( stack: stackChar)[0])) {
207             // Пушим функцию в результирующий стек
208             s21_push_stack_char( stack: stackResult,  symbols: s21_pop_stack_char( stack: stackCha
209           }
210         }
211       }
212       //При добавлении строки, мы создаем новую строку внутри стека, поэтому эту чистим, что
213       free(tmp_op);
214     }
215   }
```

s21_parser.c

```c
// функция определения приоритета операции
int s21_get_priority(char *opr) {
  int res;
  if (opr != NULL) {
    switch (*opr) {
    case '-':
    case '+':
      res = 1;
      break;
    case '~':
    case '*':
    case '/':
    case 'm':
      res = 2;
      break;
    case '^':
      res = 3;
      break;
    case ')':
    case ',':
      res = 4;
      break;
    case '(':
      res = 5;
      break;
    default:
      res = -1;
      break;
    }
  } else {
    res = -1;
  }
  return res;
}

//Проверка, что функция валидная
int s21_is_fun(char *fun) {
  int res = 0;
  if (strcmp(fun, "sin") == 0 || strcmp(fun, "cos") == 0 ||
      strcmp(fun, "tan") == 0 || strcmp(fun, "acos") == 0 ||
      strcmp(fun, "asin") == 0 || strcmp(fun, "atan") == 0 ||
      strcmp(fun, "sqrt") == 0 || strcmp(fun, "ln") == 0 ||
      strcmp(fun, "log") == 0 || strcmp(fun, "x") == 0 ||
```

```c
//Проверка, что функция валидная
int s21_is_fun(char *fun) {
  int res = 0;
  if (strcmp(fun, "sin") == 0 || strcmp(fun, "cos") == 0 ||
      strcmp(fun, "tan") == 0 || strcmp(fun, "acos") == 0 ||
      strcmp(fun, "asin") == 0 || strcmp(fun, "atan") == 0 ||
      strcmp(fun, "sqrt") == 0 || strcmp(fun, "ln") == 0 ||
      strcmp(fun, "log") == 0 || strcmp(fun, "x") == 0 ||
      strcmp(fun, "Pi") == 0 || strcmp(fun, "e") == 0 ||
      strcmp(fun, "mod") == 0) {
    res = 1;
  }
  return res;
}
```

s21_stack_char.h

s21_stack_char.h

```c

#ifndef S21_STACK_CHAR_H
#define S21_STACK_CHAR_H

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

typedef struct s21_stack_char {
  char **chars;
  int size;
  int is_x;
  int is_correct;
} s21_stack_char;

void s21_copy_stack(s21_stack_char stack_src, s21_stack_char *stack_dst);
void s21_init_stack_char(s21_stack_char *stack);
void s21_push_stack_char(s21_stack_char *stack, char *symbols);
void s21_remove_stack_char(s21_stack_char *stack);
void s21_ERROR(s21_stack_char *stack);
char *s21_pop_stack_char(s21_stack_char *stack);
char *s21_show_top_char(s21_stack_char *stack);
int s21_is_empty_stack_char(s21_stack_char stack);
int s21_is_fun(char *fun);
void s21_reverse_stack_char(s21_stack_char *stack);

#endif // S21_STACK_CHAR_H

```

s21_stack_char.c

s21_stack_char.h | s21_stack_char.c

```c
#include "s21_stack_char.h"

//Инициализация стека
void s21_init_stack_char(s21_stack_char *stack) {
    stack->chars = (char **) calloc( count: 256, size: sizeof(char));
    stack->size = 0;
    stack->is_x = 0;
    stack->is_correct = 1;
}

//Добавления нового значения в стек
void s21_push_stack_char(s21_stack_char *stack, char *symbols) {
    stack->chars[(stack->size)++] = symbols;
}

//Проверка на пустоту стека
int s21_is_empty_stack_char(s21_stack_char stack) {
    char res = 0;
    if (stack.size <= 0)
        res++;
    return res;
}


//Изъятие верхнего значения в стеке
char *s21_pop_stack_char(s21_stack_char *stack) { // присваивать и потом чистить
    char *res;
    if (stack->size == 0) {
        res = NULL;;
    } else {
        res = stack->chars[--(stack->size)];
    }
    return res;
}


//Просмотр верхнего значения стека
char *s21_show_top_char(s21_stack_char *stack) {
    char *res;
    if (s21_is_empty_stack_char(*stack)) {
        res = NULL;
    } else {
        res = stack->chars[stack->size - 1];
```

s21_stack_char.c

```c
    }
    return res;
}


//Очистка стека
void s21_remove_stack_char(s21_stack_char *stack) {
    for (int i = 0; i < stack->size; ++i) {
        free(stack->chars[i]);
    }
    stack->size = 0;
    stack->is_x = 0;
    free(stack->chars);
}


//Переворачивание стека
void s21_reverse_stack_char(s21_stack_char *stack) {
    for (int i = 0, j = stack->size - 1; i < j; ++i, --j) {
        char *tmp = stack->chars[i];
        stack->chars[i] = stack->chars[j];
        stack->chars[j] = tmp;
    }
}

//Копирование стека
void s21_copy_stack(s21_stack_char stack_src, s21_stack_char *stack_dst) {
    stack_dst->size = stack_src.size;
    stack_dst->chars = (char **) calloc( count: stack_dst->size,  size: sizeof(char *));
    stack_dst->is_x = stack_src.is_x;
    for (int i = 0; i < stack_dst->size; ++i) {
        stack_dst->chars[i] =
                (char *) calloc( count: strlen( s: stack_src.chars[i]),  size: sizeof(char));
        strcpy(stack_dst->chars[i], stack_src.chars[i]);;
    }
    stack_dst->is_correct = stack_src.is_correct;
}

//Указание на ошибку
void s21_ERROR(s21_stack_char *stack) { stack->is_correct = 0; }
```

s21_stack_node.h

s21_stack_node.h

```h
#ifndef S21_STACK_NODE_H
#define S21_STACK_NODE_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

enum {
    NUM = 1,
    OPR = 2,
    FUN = 3,
    UNO = 4
};

typedef struct s21_node {
    int type;
    char *val;
    struct s21_node *left;
    struct s21_node *right;
} s21_node;

typedef struct s21_stack_node {
    s21_node **nodes;
    int size;
} s21_stack_node;

void s21_init_stack_node(s21_stack_node *stack);
void s21_push_stack_node(s21_stack_node *stack, char *str);
void s21_remove_stack_node(s21_stack_node *stack);
s21_node *s21_pop_stack_node(s21_stack_node *stack);
int s21_is_empty_stack_node(s21_stack_node stack);
void s21_show_tree(s21_node *tree, int i);
void s21_remove_tree(s21_node **tree);
void s21_prime(s21_node **tree);
char *reverse(s21_node *tree);


#endif // S21_STACK_NODE_H

```

s21_stack_node.c

s21_stack_node.c

```c
#include "s21_stack_node.h"

//Инициализация дерева выражений
void s21_init_stack_node(s21_stack_node *stack) {
    stack->size = 0;
    stack->size = 0;
    stack->nodes = (s21_node **) calloc( count: 256, size: sizeof(s21_node *));
}


//Добавления нового значения в дерево выражений
void s21_push_stack_node(s21_stack_node *stack, char *val) {
    //Создаем новую ноду
    s21_node *tmp = (s21_node *) malloc( size: sizeof(s21_node));
    tmp->val = val;
    //Если добавляемая нода - число
    if (isdigit( c: val[0]) || *val == 'e' || *val == 'P') {
        tmp->type = NUM;
        tmp->left = NULL;
        tmp->right = NULL;
    // Если добавляемая нода - функция
    } else if (isalpha( c: val[0]) && val[0] != 'm') {
        tmp->type = FUN;
        tmp->right = s21_pop_stack_node(stack);
    // Если добавляемся нода - оператор и не унарный минус
    } else if (val[0] != '~') {
        tmp->type = OPR;
        tmp->right = s21_pop_stack_node(stack);
        tmp->left = s21_pop_stack_node(stack);
    // Если добавляемая ноад унарный минус
    } else {
        tmp->type = UNO;
        tmp->right = s21_pop_stack_node(stack);
    }
    //Добавляем в дерево новую ноду
    stack->nodes[(stack->size)++] = tmp;
}


//Очистка дерева
void s21_remove_stack_node(s21_stack_node *stack) {
    stack->size = 0;
    free(stack->nodes);
```

s21_stack_node.c

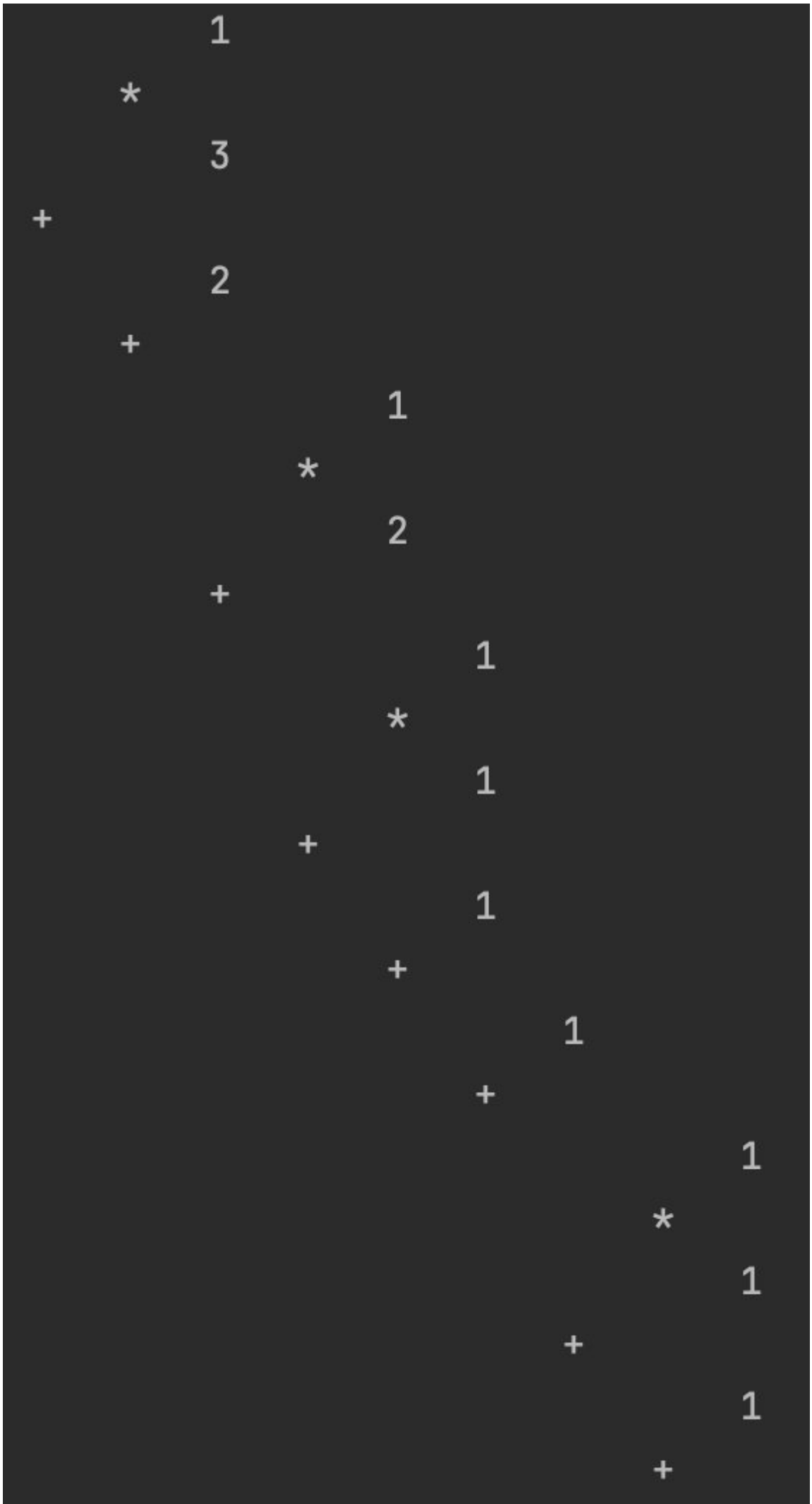
```c
}

//Извлечение значения из дерева выражений
s21_node *s21_pop_stack_node(s21_stack_node *stack) {
    if (stack->size == 0) {
        return NULL;
    } else {
        return stack->nodes[--(stack->size)];
    }
}

//Проверка на пустоту
int s21_is_empty_stack_node(s21_stack_node stack) {
    char res = 0;
    if (stack.size == 0)
        res++;
    return res;
}

//Отображение дерева
void s21_show_tree(s21_node *tree, int i) {
    if (i == 0) printf("\n\n\n");
    if (tree->right != NULL) {
        s21_show_tree( tree: tree->right,  i: i + 1);
    }
    for (int j = 0; j < i; ++j) {
        printf("\t");
    }
    printf("%s\n", tree->val);
    if (tree->left != NULL) {
        s21_show_tree( tree: tree->left,  i: i + 1);
    }
}

//Удаление дерева
void s21_remove_tree(s21_node **tree) {
    if (*tree != NULL) {
        if ((*tree)->right != NULL) {
            s21_remove_tree( tree: &(*tree)->right);
        }
        if ((*tree)->left != NULL) {
            s21_remove_tree( tree: &(*tree)->left);
        }
```

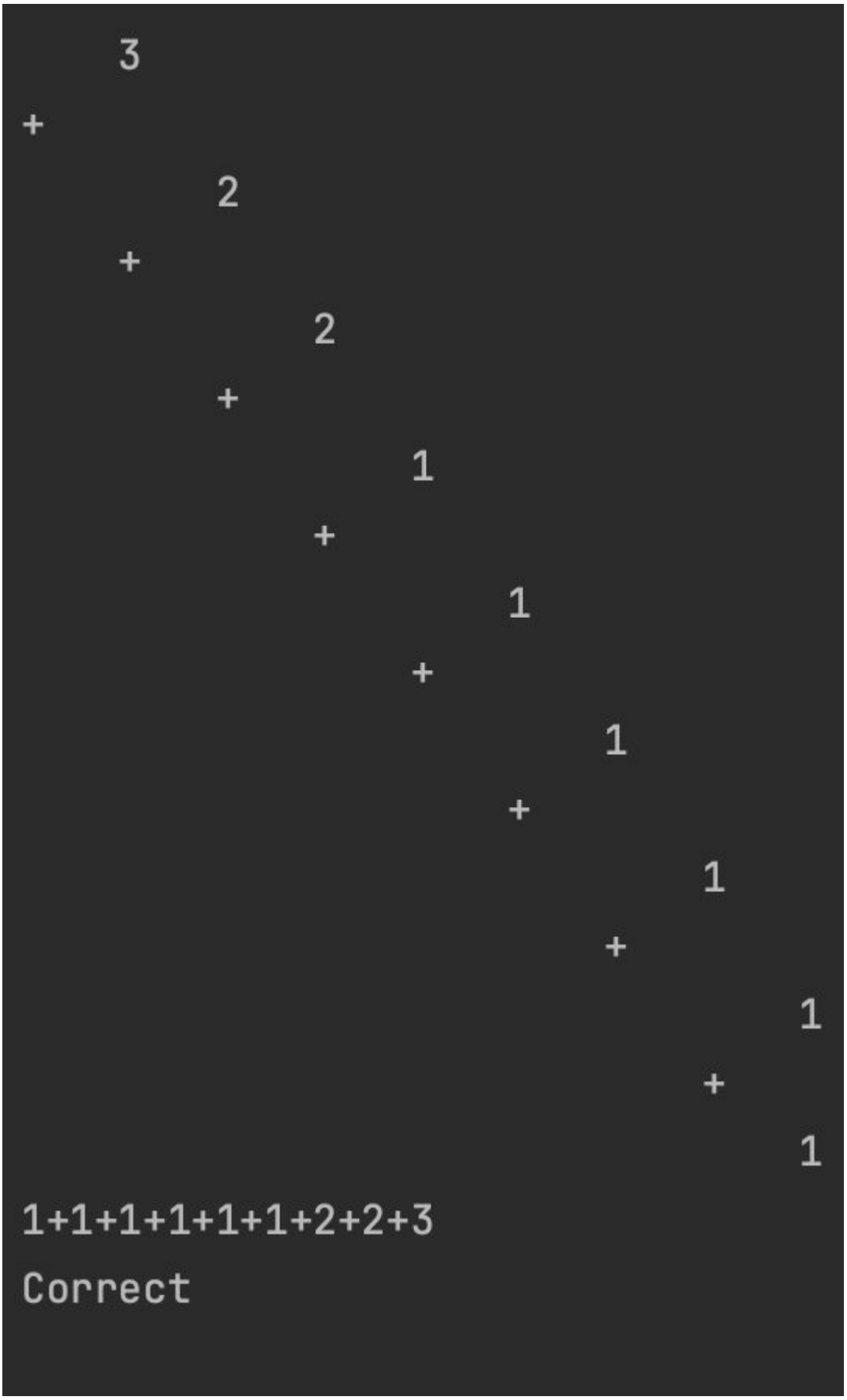
s21_stack_node.c

```
            free(*tree);
        }
    }


    //Убираем умножение на 1
    void s21_prime(s21_node **tree) {
        if ((*tree)->right != NULL) {
            s21_prime( tree: &(*tree)->right);
        }
        if ((*tree)->left != NULL) {
            s21_prime( tree: &(*tree)->left);
        }
        if ((*tree)->val[0] == '*') {
            if ((*tree)->right->val[0] == '1' && strlen( s: (*tree)->right->val) == 1) {
                s21_remove_tree( tree: &((*tree)->right));
                s21_node *left = (*tree)->left;
                free(*tree);
                (*tree) = left;
            } else if ((*tree)->left->val[0] == '1' && strlen( s: (*tree)->left->val) == 1) {
                s21_remove_tree( tree: &((*tree)->left));
                s21_node *right = (*tree)->right;
                free((*tree));
                (*tree) = right;
            }
        }
    }

    //Перевод из дерева в вид выражения
    char *reverse(s21_node *tree) {
        char *res = calloc( count: 256,  size: sizeof(char));

        if (tree->left != NULL) {
            strcat(res, reverse( tree: tree->left));
        }

        strcat(res, tree->val);

        if (tree->right != NULL) {
            strcat(res, reverse( tree: tree->right));
        }

        return res;
```

Вввод:

```
1+1+1*1+1+1+1+1*1+2*1+2+3*1
```


1
*
3
+
2
+
1
*
2
+
1
*
1
+
1
+
1
*
1
+
1


3
+
2
+
2
+
1
+
1
+
1
+
1
+
1
+
1
1+1+1+1+1+1+2+2+3
Correct

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 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:** Я научился работать со сложной структурой программы, делать свои библиотеки и структуры и объединять их в один проект, также я научился создавать сложные проекты и алгоритмы, в частности алгоритм Дейкстры по переводу строчного выражения в обратную польскую нотацию, её в свою очередь переводить в древовидный вид и проводить манипуляции уже с ней, писать отчеты, делать скриншоты, комментировать код

Недочёты при выполнении задания могут быть устранены следующим образом: _____

Подпись студента _____