МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Московский авиационный институт

(национальный исследовательский университет)

Институт № 8

Информационных технологий и прикладной математики

Кафедра 813 «Компьютерная математика»

КУРСОВАЯ РАБОТА

по дисциплине «Фундаментальная информатика»

на тему: Разработка программного обеспечения
«MATH INTERPRETER»

Выполнил: студент группы М8О-110Б-20

Нягин Анатолий Сергеевич

(Фамилия, имя, отчество)

(подпись)

Принял: доцент кафедры 813

Егорова Евгения Кирилловна

(Фамилия, имя, отчество)

(подпись)

Оценка: **Дата**:

Москва, 2020

Содержание

1 Техническое задание

Интерпретатор математических выражений

- Распознавание математического выражения с операциями +, –, ×, ÷, ^.
- Проверка корректности ввода, в том числе корректной расстановки скобок.
- Поддержка добавления и использования однобуквенных переменных (сохранение после работы в файл).
- Загрузка при старте переменных из файла.
- Очистка переменных.

1.1 Функциональные возможности приложения

Данное приложение способно вычислять значения любых математических выражений с операциями +, –, ×, ÷, ^. Имеется возможность определять однобуквенные переменные (всего 52: a..z,A..Z), а после использовать определенные переменные в математических выражениях. Определенные переменные можно сохранить в файл с помощью соответствующей команды save. Можно запустить команды интерпретатора из другого файла с помощью соответствующей команды run. Можно узнать краткую справку по возможностям интерпретатора и примерами ввода с помощью команды info. Команда exit завершает работу интерпретатора с очисткой выделенной под программу памяти, а определенные переменные записывает в файл с именем "ProgramVarsAfterExit.txt". Входные данные всегда являются командами от пользователя в терминал. При желании, можно запустить команды из файла, но для этого нужно в терминале в запущенном интерпретаторе написать команду run с именем запускаемого файла.

2 Руководство пользователя

Запуск и все взаимодействие с программой производится в терминале.

После запуска приложение выводит название ПО, заключенное в рамку, и кратчайшую справку по командам info и exit. После идут символы >>>, являющиеся

приглашением к вводу математического выражения или команды. Пустая строка или строка состоящая из пробелов вызывают ожидание новой команды. Пробелы в начале команды игнорируются интерпретатором. Максимальная длина команды - 200.

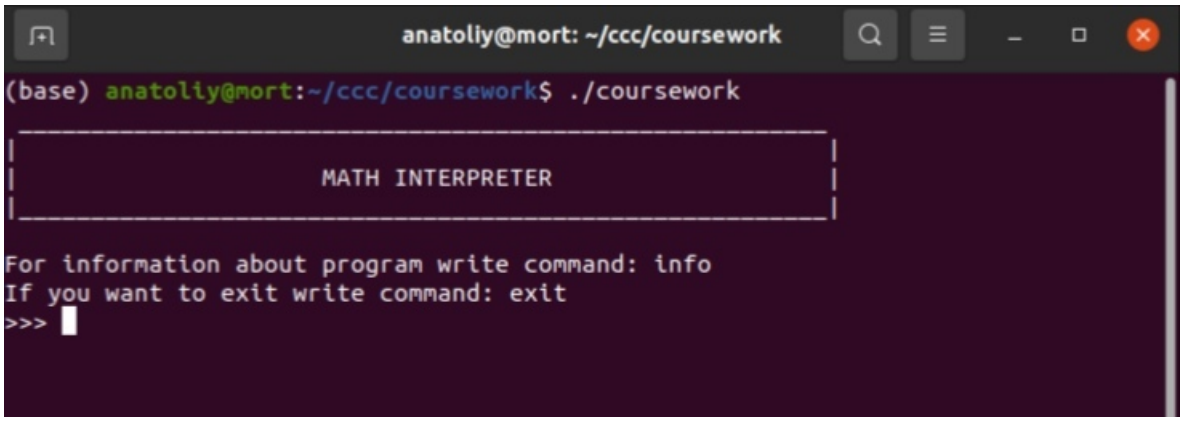

Рис. 1: Запуск ПО.

Интерпретатор принимает на вход математическое выражение с числами, переменными, знаками операций и скобками и выводит на экран значение введённого выражения. Список допустимых операций: $+$, $-$, $\times$, $\div$, $\hat{}$. Интерпретатор принимает числа со значащими нулями. Если число дробное, то перед и после точки должна быть цифра.

Можно определить однобуквенную переменную. Для этого требуется ввести команду в формате: <var_name> = <math_expression>. В переменную сохранится значение вычисленного выражения. После ввода можно использовать данные переменные в математических выражениях. При вводе названия переменной выводится её значение.

Переменные можно сохранить в файл с помощью команды save <file_name>. Перед ключевым словом save и именем файла пробела может и не быть. Считывание имени файла начинается с первого после ключевого save непробельного символа и до конца команды. Если в директории ПО нет файла с таким именем, то создастся новый файл, в который построчно запишутся переменные в формате <var_name> = <var_value>. Если файл существует, то данные переменных перезапишутся.

Команда run <file_name> запускает выполнение файла с записанными в нём построчно выражениями. Пробел между командой и именем файла может быть пропущен. Считывание имени файла начинается с первого после ключевого run непробельного символа и до конца команды.

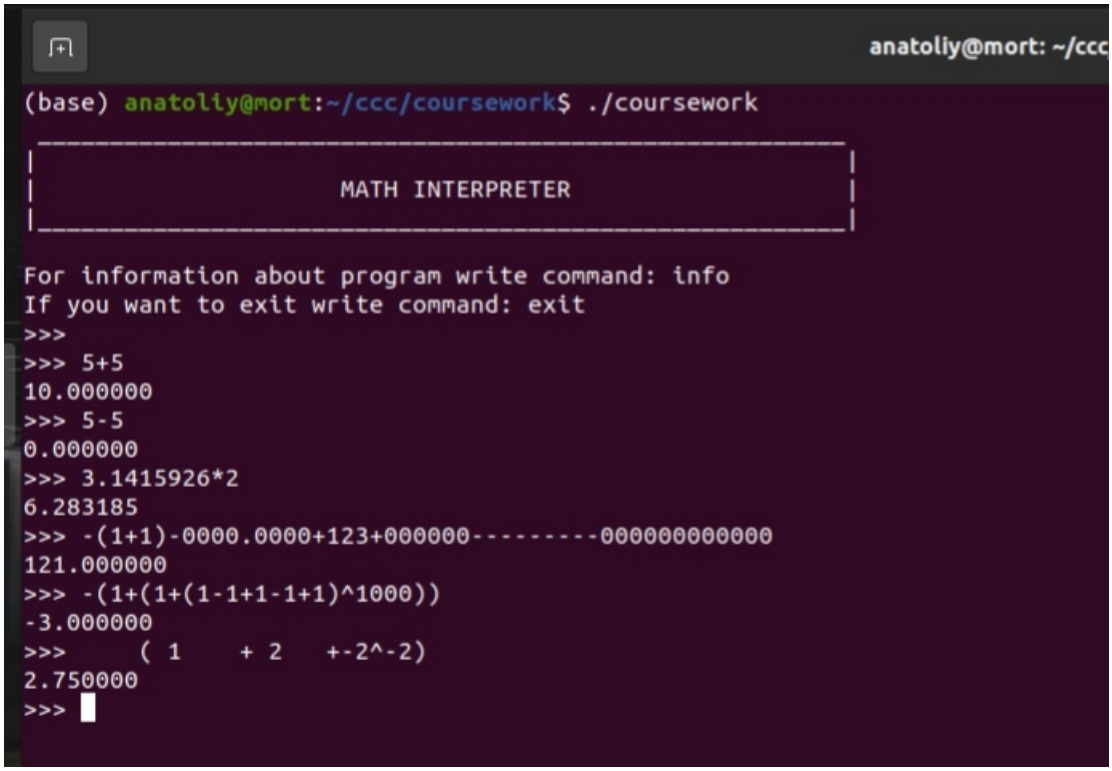

Рис. 2: Результаты вычислений разных математических выражений.

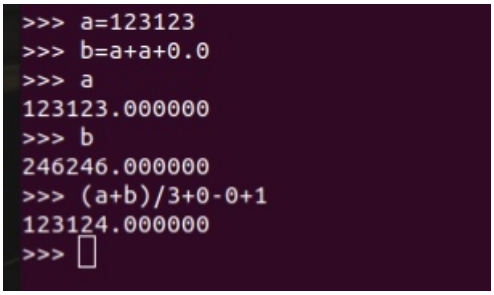

Рис. 3: Переменные.

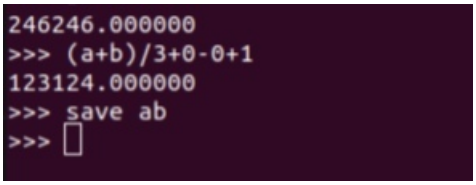

Рис. 4: Команда save сохраняет переменные в файл «ab».

Для получения информации об использовании интерпретатора используется команда info. После и перед командой не должно быть непробельных символов, пробелы игнорируются.

Рис. 5: Содержимое файла «ab».

```
>>> save ab
>>> a = 0
>>> b = 0
>>> a
0.000000
>>> b
0.000000
>>> run ab
>>> a
123123.000000
>>> b
246246.000000
>>>
```

Рис. 6: Запуск сохранённого файла, содержащего переменные.

```
>>>
>>>
>>>
>>>
>>> info
Coursework of Nyagin Anatoliy, group M8O-110B-20
This mathematical interpreter can calculate the values of mathematical expressions.
To do this, write a mathematical expression in the console after >>>
Operations +, -, *, /, ^ are supported.
For example, >>> -1+1

It is also possible to define single-letter variables.
To do this, write the name of the variable, sign =, the value of the variable (mathematical expression).
For example, >>> a = (b + 2)*-2

You can find out the value of a particular variable by writing its name.
For example, >>> a

You can save the values of variables defined during the program using the command: save <file_name>
Then the values of the variables will be saved in a file <file_name> in the directory containing the program.
For example, >>> save VarValues.txt

You can execute pre-recorded commands from a file with the command: run <file_name>
File <file_name> must be stored in the same directory where the program is stored.
For example, >>> run linesforprog.txt

To turn off the program, write the command: exit
For example, >>> exit
>>>
```

Рис. 7: Команда info дает краткую инструкцию к интерпретатору.

Для завершения работы программы используется команда exit. После и перед командой не должно быть непробельных символов, пробелы игнорируются. Перед завершением интерпретатора все определенные во время работы переменные записываются

в файл «ProgramVarsAfterExit.txt», а вся выделенная память под программу очищается.

Рис. 8: Команда exit закрыла программу и сохранила определенные переменные в файл «ProgramVarsAfterExit.txt».

Рис. 9: Содержимое файла «ProgramVarsAfterExit.txt».

Ввод некорректного выражения или команды приводит к сообщению об ошибке. Всего обрабатывается 21 видов ошибок. Если это возможно, выводится индекс места в строке команды, в которой произошла ошибка.

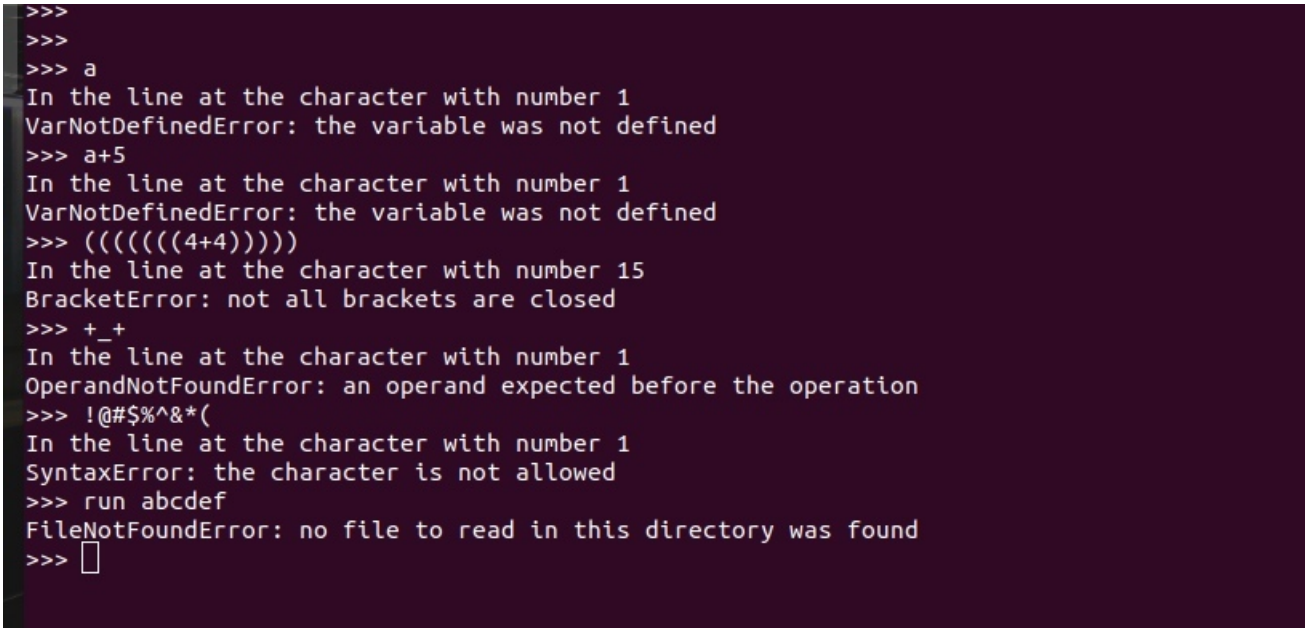

Рис. 10: Реакция программы на некорректный ввод данных.

3 Структура приложения

Проект разделен на 5 основных модулей:

1. main — модуль, отвечающий за прием ввода от пользователя, вывода имени ПО и краткой справки;
2. interpreter — модуль, отвечающий за распознавание команд и их выполнение;
3. mathexpression — модуль, отвечающий за вычисление значения математического выражения;
4. errors — модуль, в котором определена функция вывода ошибки;
5. stack — модуль, в котором определена структура данных стек.

3.1 Модуль main

В main.c находится часть, отвечающая за прием ввода от пользователя, вывода имени ПО и краткой справки. Введеная пользователем строка подается на вход функции do_command_line, реализованной в interpreter.c. При некорректности введенной строки запускается функция print_error из errors.c. В main.h записаны подключения всех заголовочных файлов, требуемых для работы всей программы.

3.2 Модуль interpreter

Функции модуля interpreter:

- void print_info() — выводит краткую справку по программе;
- void save_vars_to_file(const char file_name[], const Variables *vars) — сохраняет определенные переменные из структуры по указателю vars,в файл <file_name> в формате <var_name> = <value> построчно;
- Pair run_commands_from_file(const char file_name[], Variables *vars) — запускает команды из файла <file_name>.Команды, которые меняют значения переменных, меняют переменныеиз структуры по указателю vars.В случае успеха, возвращается структура Pair с полем number равным 0.Иначе, функция обрывается на той строке файла, в которой допущена ошибка.В поток stderr пишется в какой строке произошла ошибка. Возвращаетсяструктура Pair с полем number, равным номеру ошибки, и с полем index,равным индексу места в строке файла, в которой произошла ошибка;
- Pair do_command_line(const char line[], Variables * vars, double *result, const char program_name[]) — распознает и выполняет команду интерпретатора из строки line. Если в команде присутствуют переменные, они берутся из структуры по указателю vars. Если команда представляет собой только математическое выражение, то в случае его корректности сохраняет его значение по указателю result. В program_name дается имя файла, из которого запускаются команды. В случае успеха возвращается структура Pair с полем number, равным 0. Иначе возвращается структура Pair с полем number, равным номеру ошибки, и с полем index, равным индексу места в строке line, в которой произошла ошибка;

3.3 Модуль mathexpression

Функции модуля mathexpression:

- double get_nan() — возвращает nan;
- void do_vars_empty(Variables *vars) — делает все значения переменных в структуре, переданной по указателю - nan (неопределенными);

- int is_only_binary_oper(char ch) — возвращает 1, если поданный в нее аргумент один из символов +,–,×,÷,^, иначе 0;
- int is_oper(char ch) — возвращает 1, если поданный в нее аргумент является символом операции, иначе 0;
- int maybe_is_correct_sym(char ch) — возвращает 1, если поданный в нее аргумент является допустимым в математическом выражении, иначе 0;
- double operate(double arg1, char operation, double arg2) — принимает аргументы, операцию и возвращает результат операции к аргументам;
- int priority(char operation) — принимает символ операции и возвращает ее приоритет (1, 2, 3);
- int var_index(char var) — принимает символ (имя переменной) и возвращает индекс в массиве;
- double var_value(char var, const Variables * vars) — принимает символ (имя переменной) и возвращает ее значение;
- Pair expression_value(const char expression[], const Variables *vars,double *result) — вычисляет значение математического выражения expression, в котором могут находиться однобуквенные переменные из структуры по указателю vars. Если математическое выражение корректно, то по указателю result записывается значение выражения и возвращается структура Pair с полем number равным 0. Иначе result не меняется, а функция возвращает Pair с полем number равным номеру ошибки и с полем index равным индексу места в строке expression, в котором произошла ошибка;

Структуры модуля mathexpression

- Variables:
  - double val[52] — массив однобуквенных переменных;
- Pair:
  - int number — номер ошибки;
  - int index — номер места в строке, в которой произошла ошибка;

3.4 Модуль errors

Функции модуля errors:

- void print_error(Pair pairError) — выводит информацию об ошибке в поток stderr. Принимает структуру Pair с полем number равным номеру ошибки и полем index равным индексу места в строке, в которой произошла ошибка.

3.5 Модуль stack

Функции модуля stack:

- Stack* create_stack() — выделяет на куче память для структуры Stack и возвращает указатель на эту область памяти;
- int delete_stack(Stack **stack) — принимает указатель на указатель на структуру Stack и освобождает память, выделенную под нее. В случае успеха вовзращает 0, иначе 1;
- int push(Stack *stack, Elem data) — принимает указатель на структуру Stack и значение типа Elem и ставит значение на вершину стека. В случае успеха возвращает 0, иначе 1;
- int pop(Stack *stack, Elem *to_save_here) — принимает первым аргументом указатель наструктуру Stack. Если указатель NULL, то возвращает 1, еслиструктура пуста, то возвращает 2. В случае успеха возвращает 0,а по указателю данному вторым аргументом сохраняетсязначение с вершины стека, а сам элемент с вершины стека удаляется;
- int top(Stack *stack, Elem * to_save_here) — принимает первым аргументом указатель наструктуру Stack. Если указатель NULL, то возвращает 1, если структура пуста, то возвращает 2. В случае успеха возвращает 0, а по указателю данному вторым аргументом сохраняется значение с вершины стека;

Структуры и объединения модуля stack:

- Elem — объединение, которое может хранить значение типа double и char;
- Node — узел стека:

  - Elem value — значение в узле;
  - struct *Node next — указатель на следующий узел;

- Stack:
  - Node *head — указатель на вершину стека;

3.6 Последовательность вызовов функций

В main вызывается функция do_command_line. В зависимости от того, какая команда была передана в функцию, do_command_line вызывает разные функции.

- Если команда содержала ошибку, то возвращается PairError с номером ошибки. Далее в main вызывается print_error(PairError).
- Если команда представляла собой команду info, то вызывается функция printf_info. do_command_line возвращает pairError с номером ошибки равным 0.
- Если команда представляла собой команду exit, то do_command_line возвращает pairError с номером, соответствующим коду завершения программы. Далее в main происходит завершение работы приложения.
- Если команда представляла собой save с именем файла, то вызывается функция save_vars_to_file. После работы save_vars_to_file функция do_command_line возвращает pairError с номером ошибки равным 0.
- Если команда представляла собой run с именем файла, то вызывается функция run_commands_from_file. Далее в функции run_commands_from_file к строкам файла рекурсивно вызывается функция do_command_line. После завершения run_commands_from_file функция do_command_line возвращает pairError с номером ошибки равным 0.
- Если команда представляла собой математическое выражение, то к нему вызывается функция expression_value. Про ее работу подробнее изложено в разделе «Основные алгоритмы». Если в выражении оказалась ошибка, то возвращается pairError с номером ошибки, иначе по указателю, переданным аргументом, сохраняется значение математического выражения, а функция expression_value

возвращает pairError с номером ошибки равным 0. Далее do_command_line возвращает полученный ранее pairError.

- Если команда представляла собой выражение вида <var> = <math_expression>, то к части <math_expression> вызывается функция expression_value.

4 Основные алгоритмы

Основным используемым алгоритмом является алгоритм сортировочной станции Дейкстры для вычисления математического выражения в функции expression_value.

Алгоритм:

Стек символов;

Стек чисел.

Пока строка не пустая:

Если объект - число, то кладем на вершину стека чисел.

Если объект - открывающая скобка, то кладем на вершину стека символов.

Если объект - операция, то

Если стек символов пуст, то кладем объект на вершину стека символов.

Если символ на вершине стека символов является открывающей скобкой, то кладем объект на вершину стека символов.

Если символ на вершине стека является операцией, то

Если объект - возведение в степень и символ с вершины стека - унарный минус, то кладем объект на стек символов.

Если объект - возведение в степень и символ с вершины стека - возведение в степень, то кладем объект на стек символов.

Если объект - унарный минус и символ с вершины стека - возведение в степень, то кладем объект на стек символов.

Если объект - унарный минус и символ с вершины стека - унарный минус, то кладем объект на стек символов.

Если приоритет объекта строго больше приоритета символа с вершины стека, то кладем объект на стек символов.

Иначе

Пока (стек символов не пуст) и (символ с вершины стека символов - операция) и (приоритет символа с вершины стека строго больше приоритета объекта)

Взять число со стека чисел. (первое число)

Если символ со стека символов - унарный минус, то снять символ со стека символов, применить к первому числу и положить обратно на стек чисел.

Иначе взять число со стека чисел (второе число), снять символ со стека символов, применить операцию ко второму числу и первому в данном порядке и положить результат на стек чисел.

Если объект - закрывающая скобка, то

Взять число со стека чисел. (первое число)

Если символ со стека символов - унарный минус, то снять символ со стека символов, применить к первому числу и положить обратно на стек чисел.

Иначе взять число со стека чисел (второе число), снять символ со стека символов, применить операцию ко второму числу и первому в данном порядке и положить результат на стек чисел.

Пока стек символов не пуст:

Взять число со стека чисел. (первое число)

Если символ со стека символов - унарный минус, то снять символ со стека символов, применить к первому числу и положить обратно на стек чисел.

Иначе взять число со стека чисел (второе число), снять символ со стека символов, применить операцию ко второму числу и первому в данном порядке и положить результат на стек чисел.

5 Список использованных источников

1. Дейтел Х., Дейтел П. Как программировать на Си. – М.: Бином, 2000. – 994 с

2. Керниган Б. В. Язык программирования C, 2-е издание. – Издательский дом Вильямс, 2012. — 702 с

6 Приложение

6.1 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "main.h"


int main(char argc, char* argv[]) {
        printf(" _____________________________________________________\n");
        printf("|                                                     |\n");
        printf("|                  MATH INTERPRETER                   |\n");
        printf("|_____________________________________________________|\n");
        printf("\nFor information about program write command: info\n");
        printf("If you want to exit write command: exit\n");
        char input[200] = {0};
        Variables vars;
        do_vars_empty(&vars);

        double result;
        while (1) {
            fgets(input, 1, stdin);
            printf(">>> ");
            fgets(input, 200, stdin);

            result = get_nan();
            Pair pairError = do_command_line(input, &vars, &result, argv[0]);
            if (pairError.number == EXIT_NUMBER) {
                return 0;
            }
```

```
            if (pairError.number) {
                print_error(pairError);
            }

            else {
                if (!isnan(result)) {
                    printf("%lf\n", result);
                }
            }
            memset(input, '\0', sizeof(input));
        }
        return 0;
}
```

6.2 main.h

```
#include "stack.h"
#include "mathexpression.h"
#include "errors.h"
#include "interpreter.h"
```

6.3 interpreter.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "mathexpression.h"
#include "errors.h"
#include "interpreter.h"
#include "errors.h"

void printf_info() {
```

```
    // Функция выводит краткую справку по программе
    printf("Coursework of Nyagin Anatoliy, group M8O-110B-20\n");
    printf("This mathematical interpreter can calculate the values \
    of mathematical expressions.\n");
    printf("To do this, write a mathematical expression in the \
    console after >>>\n");
    printf("Operations +, -, *, /, ^ are supported.\n");
    printf("For example, >>> -1+1\n\n");

    printf("It is also possible to define single-letter variables.\n");
    printf("To do this, write the name of the variable, sign =, the\
    value of the variable (mathematical expression).\n");
    printf("For example, >>> a = (b + 2)*-2\n\n");

    printf("You can find out the value of a particular variable by \
    writing its name.\n");
    printf("For example, >>> a\n\n");

    printf("You can save the values of variables defined during the\
    program using the command: save <file_name>\n");
    printf("Then the values of the variables will be saved in a file\
    <file_name> in the directory containing the program.\n");
    printf("For example, >>> save VarValues.txt\n\n");

    printf("You can execute pre-recorded commands from a file with\
    the command: run <file_name>\n");
    printf("File <file_name> must be stored in the same directory \
    where the program is stored.\n");
    printf("For example, >>> run linesforprog.txt\n\n");

    printf("To turn off the program, write the command: exit\n");
    printf("For example, >>> exit\n");
}
```

```c
void save_vars_to_file(const char file_name[], const Variables *vars) {
    /*Сохраняет определенные переменные из структуры по указателю vars,
    в файл <file_name> в формате <var_name> = <value> построчно*/

    FILE * file = fopen(file_name, "w");
    int i=0;
    for (i=0; i < 26; i++) {
        if (!isnan(vars->val[i])) {
            fprintf(file, "%c = %lf\n", i+'a', vars->val[i]);
        }
    }

    for (i=26; i < 52; i++) {
        if (!isnan(vars->val[i])) {
            fprintf(file, "%c = %lf\n", i+'A'-26, vars->val[i]);
        }
    }
    fclose(file);
}


Pair run_commands_from_file(const char file_name[], Variables *vars) {
    /*Функция, которая запускает команды из файла <file_name>.
    Команды, которые меняют значения переменных, меняют переменные
    из структуры по указателю vars.
    В случае успеха, возвращается структура Pair с полем number равным 0.
    Иначе, функция обрывается на той строке файла, в которой допущена ошибка.
    В поток stderr пишется в какой строке произошла ошибка. Возвращается
    структура Pair с полем number, равным номеру ошибки, и с полем index,
    равным индексу места в строке файла, в которой произошла ошибка.
    */
```

```
    FILE * file = fopen(file_name, "r");
    Pair pairError;
    pairError.number = 0;
    if (file == NULL) {
        pairError.number = FILE_NOT_FOUND_ERROR;
        return pairError;
    }
    int line_index=1;
    char file_line[200]={0};
    double result;
    for (line_index=1; !feof(file); line_index++) {
        result = get_nan();
        fgets(file_line, 200, file);
        pairError = do_command_line(file_line, vars, &result, file_name);
        if (pairError.number == EXIT_NUMBER) {
            fclose(file);
            return pairError;
        }
        if (pairError.number) {
            fprintf(stderr, "In <%s> file in line number %d\n",
            file_name, line_index);
            fclose(file);
            return pairError;
        }
        if (!isnan(result)) {
            printf("Line %d - %lf\n", line_index, result);
        }

    }
    fclose(file);
    return pairError;
}
```

```
Pair do_command_line(const char line[], Variables * vars, double *result,
    const char program_name[]) {
   /*Функция распознает и выполняет команду интерпретатора из строки line.
    Если в команде присутствуют переменные, они берутся из структуры по
    указателю vars. Если команда представляет собой только математическое
    выражение, то в случае его корректности сохраняет его значение по
    указателю result. В program_name дается имя файла, из которого
    запускаются команды. В случае успеха возвращается структура Pair с
    полем number, равным 0. Иначе возвращается структура Pair с полем
    number, равным номеру ошибки, и с полем index, равным индексу места
    в строке line, в которой произошла ошибка.
    */


    int i=0;
    for (i = 0; line[i] == ' ';) {
        i++;
    }

    Pair pairError;
    pairError.number = 0;
    pairError.index = i+1;

    if (line[i+1] == '\0') {
        return pairError;
    }

    if (!(maybe_is_correct_sym(line[i]) || (line[i] == '='))) {
        pairError.number = CHAR_IS_NOT_ALLOWED_ERROR;
        return pairError;
    }
```

```
if (isalpha(line[i])) {
    // keyword or expression A=B or math expression

    if (isalpha(line[i+1])) {
        // is keyword

        char file_name[30]={0};
        int j = 0;
        if (strncmp(line+i, "exit", 4) == 0) {
            for (i=i+4; line[i+1];i++) {
                if (line[i] != ' ') {
                    pairError.index = i+1;
                    pairError.number =
                    AFTER_COMM_NOT_EXPECTED_CHARS_ERROR;
                    return pairError;
                }
            }
            save_vars_to_file("ProgramVarsAfterExit.txt", vars);
            pairError.number = EXIT_NUMBER;
            return pairError;
        }

        if (strncmp(line+i, "info", 4) == 0) {
            for (i=i+4; line[i+1];i++) {
                if (line[i] != ' ') {
                    pairError.index = i+1;
                    pairError.number =
                    AFTER_COMM_NOT_EXPECTED_CHARS_ERROR;
                    return pairError;
                }
            }
            printf_info();
            pairError.number = 0;
```

```
        return pairError;
    }

    if (strncmp(line+i, "save", 4) == 0) {
        for (i=i+4; line[i] == ' ';) {
            i++;
        }
        for (; line[i+1]; i++) {
            file_name[j] = line[i];
            j++;
        }
        if (file_name[0] == '\0') {
            pairError.number = FILE_NAME_NOT_MET_ERROR;
            return pairError;
        }
        if (!strcmp(file_name, program_name)) {
            pairError.number = FILE_IS_RUNNING_ERROR;
            return pairError;
        }
        save_vars_to_file(file_name, vars);
        pairError.number = 0;
        return pairError;
    }

    if (strncmp(line+i, "run", 3) == 0) {
        for (i=i+3; line[i] == ' ';) {
            i++;
        }
        for (; line[i+1]; i++) {
            file_name[j] = line[i];
            j++;
        }
        if (file_name[0] == '\0') {
```

```
            pairError.number = FILE_NAME_NOT_MET_ERROR;
            return pairError;
        }
        if (!strcmp(file_name, program_name)) {
            pairError.number = FILE_IS_RUNNING_ERROR;
            return pairError;
        }
        pairError = run_commands_from_file(file_name, vars);
        pairError.index = i+1;
        return pairError;
    }
    // else
    pairError.number = COMMAND_NOT_EXIST_ERROR;
    return pairError;
}

else {
    if (strstr(line, "=")) {
        // A = B
        char var = line[i];
        for (i = i+1; line[i] == ' ';) {
            i++;
        }
        if (line[i] == '=') {
            double res;
            pairError = expression_value(line+i+1, vars, &res);
            if (pairError.number == 0) {
                vars->val[var_index(var)] = res;
                return pairError;
            }
            else {
                pairError.index += i;
                return pairError;
```

```
                }
            }
            else {
                pairError.number = MUST_BE_NO_CHARS_BETWEEN_ERROR;
                pairError.index = i;
                return pairError;
            }
        }
        else { //math expression
            pairError = expression_value(line, vars, result);
            if (pairError.number) {
                return pairError;
            }
        }
    }
}
if (line[i] == '=') {
    pairError.number = LEFT_PART_NOT_MET_ERROR;
    return pairError;
}
if (isdigit(line[i]) || is_oper(line[i]) || (line[i] == '(') ||
 (line[i] == ')') || (line[i] == '.')) {
    pairError = expression_value(line, vars, result);
    if (pairError.number) {
        return pairError;
    }
}
return pairError;

}
```

6.4 interpreter.h

```
void printf_info();
void save_vars_to_file(const char[], const Variables *);
Pair do_command_line(const char[], Variables *, double *, const char []);
Pair run_commands_from_file(const char[], Variables *);
```

6.5 mathexpression.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "stack.h"
#include "mathexpression.h"
#include "errors.h"

double get_nan() {
    /*Функция, возвращающая nan*/

    return -pow(-1, 0.5);
}

void do_vars_empty(Variables *vars) {
    /*Функция, делающая все значения переменных
    в структуре по указателю vars - nan (неопределенными)*/

    for (int i=0; i<52; i++) {
        vars->val[i] = get_nan();
    }
}

int is_only_binary_oper(char ch) {
```

```
    /*Функция, возвращающая 1, если ch один из символов +-/*^ ,иначе 0*/

    if ((ch == '+') || (ch == '*') || (ch == '/') || (ch == '^')) {
        return 1;
    }
    return 0;
}

int is_oper(char ch) {
    /*Функция, возвращающая 1, если ch является символом операции, иначе 0*/

    if ((ch == '+') || (ch == '-') || (ch == '*') ||
     (ch == '/') || (ch == '^')) {
        return 1;
    }
    return 0;
}

int maybe_is_correct_sym(char ch) {
    /*Функция, возвращающая 1, если ch является допустимым в
    математическом выражении, иначе 0*/

    if ((isalnum(ch)) || (is_oper(ch)) || (ch == ' ') ||
    (ch == '(') || (ch == ')') || (ch == '.')) {
        return 1;
    }
    return 0;
}


double operate(double arg1, char operation, double arg2) {
    /*Функция, выполняющая операцию operation к аргументам
    arg1 и arg2 в заданном порядке. Если operation == ~,
```

```
    то выполняет унарный минус к аргументу arg1.
    Возвращает результат, если операция одна из ~+-/*^, иначе nan*/

    if (operation == '~') // унарный минус
        return -arg1;
    if (operation == '+')
        return arg1 + arg2;
    if (operation == '-')
        return arg1 - arg2;
    if (operation == '*')
        return arg1 * arg2;
    if (operation == '/')
        return arg1 / arg2;
    if (operation == '^')
        return pow(arg1, arg2);
    return get_nan();
}

int priority(char operation) {
    /*Функция, возвращающая приоритет операции. Приоритеты
    + и -: 1
    * и /: 2
    ~ и ^: 3*/

    if ((operation == '-') || (operation == '+'))
        return 1;
    if ((operation == '*') || (operation == '/'))
        return 2;
    if ((operation == '^') || (operation == '~'))
        return 3;
    return 0;
}
```

```
int var_index(char var) {
    /*Фунция, возвращающая индекс в массиве для имени однобуквеной
    переменной var. Если var - не буква, возвращает -1*/

    if (islower(var)) {
        return (var - 'a');
    }
    if (isupper(var)) {
        return (var - 'A' + 26);
    }
    return -1;
}

double var_value(char var, const Variables * vars) {
    /*Функция возвращает значение переменной var из
    структуры по указателю vars, в случае, если
    var - буква. Иначе возвращает nan*/

    if (var_index(var) != -1) {
        return vars->val[var_index(var)];
    }
    return get_nan();
}



Pair expression_value(const char expression[], const Variables *vars,
    double *result) {
    /*Функция, вычисляющая значение математического выражения
    expression, в котором могут находиться однобуквенные переменные
    из структуры по указателю vars. Если математическое выражение
    корректно, то по указателю result записывается значение выражения
    и возвращается структура Pair с полем number равным 0. Иначе result
```

```
не меняется, а функция возвращает Pair с полем number равным номеру
ошибки и с полем index равным индексу места в строке expression, в
котором произошла ошибка.*/


Pair pairError;
pairError.number = 0;
pairError.index = 0;

int i=0;
for (i=0; expression[i] == ' ';) {
    i++;
} // Пропускаем пробелы в начале
pairError.index = i+1;

if (expression[i+1] == '\0') {
    pairError.number = EMPTY_STRING_ERROR;
    return pairError;
}
int opened_closed_diff = 0;
// разность кол-ва открывающихся и закрывающихся скобок

char str_number[20] = {0};
// строковое представление данного числа в expression
char *after_last_digit = str_number;
// указатель на место, идущей после последней цифры
Elem element;
// данный элемент: char: буква, операция, скобка, точка, double: число
int point_number = 0;
// кол-во точек в числе

Elem top_elem; // элемент на вершине стека
Elem num2; // второй операнд
```

```
Elem num1; // первый операнд
Elem oper; // операция
Elem res; // результат подвыражения

Stack * numbers = create_stack();
Stack * symbols = create_stack();


// проверка на начало мат. выражения
if (!maybe_is_correct_sym(expression[i])) {
    pairError.number = CHAR_IS_NOT_ALLOWED_ERROR;
    delete_stack(&numbers);
    delete_stack(&symbols);
    return pairError;
}
if (expression[i] == ')') {
    pairError.number = OPENING_BRACKET_NOT_MET_ERROR;
    delete_stack(&numbers);
    delete_stack(&symbols);
    return pairError;
}
if (expression[i] == '.') {
    pairError.number = BEFORE_POINT_EXPECTED_DIGIT_ERROR;
    delete_stack(&numbers);
    delete_stack(&symbols);
    return pairError;
}
if (is_only_binary_oper(expression[i])) {
    pairError.number = BEFORE_OPERATION_EXPECTED_ERROR;
    delete_stack(&numbers);
    delete_stack(&symbols);
    return pairError;
}
```

```
if (expression[i] == '(') {
    opened_closed_diff++;

    element.sym = expression[i];
    push(symbols, element);
}

if (expression[i] == '-') {
    element.sym = '~';
    push(symbols, element);
}

if (isalpha(expression[i])) {
    element.num = var_value(expression[i], vars);
    if (isnan(element.num)) {
        pairError.number = VAR_NOT_DEFINED_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    push(numbers, element);
}

if (isdigit(expression[i])) {
    *after_last_digit = expression[i];
    after_last_digit++;
    if (!(isdigit(expression[i+1])) && expression[i+1] != '.') {
        element.num = atof(str_number);
        memset(str_number, 0, sizeof(str_number));
        push(numbers, element);
        after_last_digit = str_number;
    }
}
```

```
char prev = expression[i];
for (i=i+1; expression[i+1]; i++) {

    pairError.index = i+1;
    if (!maybe_is_correct_sym(expression[i])) {
        pairError.number = CHAR_IS_NOT_ALLOWED_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if (expression[i] == ' ') {
        continue;
    }

    if (isdigit(expression[i])) {
        if (isdigit(prev) && expression[i-1] == ' ') {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (isalpha(prev)) {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (expression[i] == ')') {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
```

```
            delete_stack(&symbols);
            return pairError;
        }
        *after_last_digit = expression[i];
        after_last_digit++;
        if (!(isdigit(expression[i+1])) && expression[i+1] != '.') {
            element.num = atof(str_number);
            memset(str_number, 0, sizeof(str_number));
            push(numbers, element);
            after_last_digit = str_number;
        }
    }

    if (isalpha(expression[i])) {
        if (prev == '.') {
            pairError.number = AFTER_POINT_EXPECTED_DIGIT_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (isdigit(prev)) {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (prev == ')') {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (isalpha(prev)) {
```

```
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        point_number = 0;

        element.num = var_value(expression[i], vars);
        if (isnan(element.num)) {
            pairError.number = VAR_NOT_DEFINED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        push(numbers, element);
    }

    if (is_only_binary_oper(expression[i])) {
        if (prev == '.') {
            pairError.number = AFTER_POINT_EXPECTED_DIGIT_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (prev == '(') {
            pairError.number = BEFORE_OPERATION_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (is_oper(prev)) {
            pairError.number = BEFORE_OPERATION_EXPECTED_ERROR;
            delete_stack(&numbers);
```

```
        delete_stack(&symbols);
        return pairError;
    }
    point_number = 0;

    if (symbols->head == NULL) {
        element.sym = expression[i];
        push(symbols, element);
        prev = expression[i];
        continue;
    }

    top(symbols, &top_elem);
    if (!(is_oper(top_elem.sym) || top_elem.sym == '~')) {
        element.sym = expression[i];
        push(symbols, element);
        prev = expression[i];
        continue;
    }

    if (expression[i] == '^') {
        if (top_elem.sym == '^' || top_elem.sym == '~') {
            element.sym = expression[i];
            push(symbols, element);
            prev = expression[i];
            continue;
        }
    }

    if (priority(expression[i]) > priority(top_elem.sym)) {
        element.sym = expression[i];
        push(symbols, element);
        prev = expression[i];
```

```
            continue;
        }
        else {
            top(symbols, &top_elem);
            while (((symbols->head != NULL) && (is_oper(top_elem.sym)
            || top_elem.sym == '~')) &&
            (priority(expression[i]) <= priority(top_elem.sym))) {
                if (top_elem.sym == '~') {
                    pop(numbers, &num1);
                    pop(symbols, &oper);
                    res.num = operate(num1.num, oper.sym, 0);
                    push(numbers, res);
                    top(symbols, &top_elem);
                    continue;
                }
                pop(numbers, &num2);
                pop(numbers, &num1);
                pop(symbols, &oper);
                if ((oper.sym == '/') && (num2.num == 0)) {
                    pairError.number = ZERO_DIVISION_ERROR;
                    delete_stack(&numbers);
                    delete_stack(&symbols);
                    return pairError;
                }
                if ((oper.sym == '^') && (num1.num <= 0) &&
                (num2.num != trunc(num2.num))) {
                    pairError.number = POWER_ERROR;
                    delete_stack(&numbers);
                    delete_stack(&symbols);
                    return pairError;
                }
                res.num = operate(num1.num, oper.sym, num2.num);
                push(numbers, res);
```

```
            top(symbols, &top_elem);
        }
        element.sym = expression[i];
        push(symbols, element);
    }
}

if (expression[i] == '-') {
    if (prev == '.') {
        pairError.number = AFTER_POINT_EXPECTED_DIGIT_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }

    if (isalnum(prev) || prev == ')') {
        // binary minus
        if (symbols->head == NULL) {
            element.sym = '-';
            push(symbols, element);
            prev = expression[i];
            continue;
        }

        top(symbols, &top_elem);
        while ((symbols->head != NULL) && (is_oper(top_elem.sym)
        || top_elem.sym == '~')) {
            if (top_elem.sym == '~') {
                pop(numbers, &num1);
                pop(symbols, &oper);
                res.num = operate(num1.num, oper.sym, 0);
                push(numbers, res);
                top(symbols, &top_elem);
```

```
                continue;
            }
            pop(numbers, &num2);
            pop(numbers, &num1);
            pop(symbols, &oper);
            if ((oper.sym == '/') && (num2.num == 0)) {
                pairError.number = ZERO_DIVISION_ERROR;
                delete_stack(&numbers);
                delete_stack(&symbols);
                return pairError;
            }
            if ((oper.sym == '^') && (num1.num <= 0) &&
            (num2.num != trunc(num2.num))) {
                pairError.number = POWER_ERROR;
                delete_stack(&numbers);
                delete_stack(&symbols);
                return pairError;
            }
            res.num = operate(num1.num, oper.sym, num2.num);
            push(numbers, res);
            top(symbols, &top_elem);
        }

        element.sym = '-';
        push(symbols, element);
    }
    if (prev == '(' || is_oper(prev)) {
        // unar minus
        if (symbols->head == NULL) {
            element.sym = '~';
            push(symbols, element);
            prev = expression[i];
            continue;
```

```
}

top(symbols, &top_elem);
if (top_elem.sym == '^' || top_elem.sym == '~' ||
!(is_oper(top_elem.sym))) {
    element.sym = '~';
    push(symbols, element);
    prev = expression[i];
    continue;
}

if (priority('~') > priority(top_elem.sym)) {
    element.sym = '~';
    push(symbols, element);
    prev = expression[i];
    continue;
}
else {
    top(symbols, &top_elem);
    while ((symbols->head != NULL) && (is_oper(top_elem.sym)
    || top_elem.sym == '~') &&
     (priority('~') <= priority(top_elem.sym))) {
        if (top_elem.sym == '~') {
            pop(numbers, &num1);
            pop(symbols, &oper);
            res.num = operate(num1.num, oper.sym, 0);
            push(numbers, res);
            top(symbols, &top_elem);
            continue;
        }
        pop(numbers, &num2);
        pop(numbers, &num1);
        pop(symbols, &oper);
```

```
                if ((oper.sym == '/') && (num2.num == 0)) {
                    pairError.number = ZERO_DIVISION_ERROR;
                    delete_stack(&numbers);
                    delete_stack(&symbols);
                    return pairError;
                }
                if ((oper.sym == '^') && (num1.num <= 0) &&
                (num2.num != trunc(num2.num))) {
                    pairError.number = POWER_ERROR;
                    delete_stack(&numbers);
                    delete_stack(&symbols);
                    return pairError;
                }
                res.num = operate(num1.num, oper.sym, num2.num);
                push(numbers, res);
                top(symbols, &top_elem);
            }
            element.sym = '~';
            push(symbols, element);
        }
    }

    point_number = 0;
}

if (expression[i] == '(') {
    if (prev == '.') {
        pairError.number = AFTER_POINT_EXPECTED_DIGIT_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if (isdigit(prev)) {
```

```
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (isalpha(prev)) {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (prev == ')') {
            pairError.number = AFTER_OPERAND_EXPECTED_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        opened_closed_diff++;
        point_number = 0;

        element.sym = expression[i];
        push(symbols, element);
    }

    if (expression[i] == ')') {
        if (prev == '(') {
            pairError.number = BRACKETS_IS_EMPTY_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if (is_oper(prev)) {
            pairError.number = AFTER_OPERATION_EXPECTED_OPERAND_ERROR;
```

```
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if (prev == '.') {
        pairError.number = AFTER_POINT_EXPECTED_DIGIT_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    opened_closed_diff--;
    if (opened_closed_diff < 0) {
        pairError.number = OPENING_BRACKET_NOT_MET_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    point_number = 0;

    top(symbols, &top_elem);
    while (top_elem.sym != '(') {
        if (top_elem.sym == '~') {
            pop(numbers, &num1);
            pop(symbols, &oper);
            res.num = operate(num1.num, oper.sym, 0);
            push(numbers, res);
            top(symbols, &top_elem);
            continue;
        }
        pop(numbers, &num2);
        pop(numbers, &num1);
        pop(symbols, &oper);
        if ((oper.sym == '/') && (num2.num == 0)) {
```

```
            pairError.number = ZERO_DIVISION_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        if ((oper.sym == '^') && (num1.num <= 0) &&
        (num2.num != trunc(num2.num))) {
            pairError.number = POWER_ERROR;
            delete_stack(&numbers);
            delete_stack(&symbols);
            return pairError;
        }
        res.num = operate(num1.num, oper.sym, num2.num);
        push(numbers, res);
        top(symbols, &top_elem);
    }
    pop(symbols, &top_elem);
}

if (expression[i] == '.') {
    if (prev == '.') {
        pairError.number = POINT_IS_EXCESS_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if (!(isdigit(prev))) {
        pairError.number = BEFORE_POINT_EXPECTED_DIGIT_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    point_number++;
```

```
            if (point_number > 1) {
                pairError.number = POINT_IS_EXCESS_ERROR;
                delete_stack(&numbers);
                delete_stack(&symbols);
                return pairError;
            }
            *after_last_digit = expression[i];
            after_last_digit++;
        }
        prev = expression[i];
    }

    if (opened_closed_diff) {
        pairError.number = NO_ALL_BRACKETS_CLOSED_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if (is_oper(expression[i-1])) {
        pairError.number = AFTER_OPERATION_EXPECTED_OPERAND_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if (expression[i-1] == '.') {
        pairError.number = AFTER_POINT_EXPECTED_DIGIT_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }

    top(symbols, &top_elem);
    while (symbols->head != NULL) {
```

```
    if (top_elem.sym == '~') {
        pop(numbers, &num1);
        pop(symbols, &oper);
        res.num = operate(num1.num, oper.sym, 0);
        push(numbers, res);
        top(symbols, &top_elem);
        continue;
    }
    pop(numbers, &num2);
    pop(numbers, &num1);
    pop(symbols, &oper);
    if ((oper.sym == '/') && (num2.num == 0)) {
        pairError.number = ZERO_DIVISION_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    if ((oper.sym == '^') && (num1.num <= 0) &&
    (num2.num != trunc(num2.num))) {
        pairError.number = POWER_ERROR;
        delete_stack(&numbers);
        delete_stack(&symbols);
        return pairError;
    }
    res.num = operate(num1.num, oper.sym, num2.num);
    push(numbers, res);
    top(symbols, &top_elem);
}

pop(numbers, &res);
*result = res.num;
delete_stack(&numbers);
delete_stack(&symbols);
```

```
    return pairError;
}
```

6.6 mathexpression.h

```
typedef struct Variables {
    double val[52];
} Variables;

typedef struct Pair {
    int number;
    int index;
} Pair;



double get_nan();
void do_vars_empty(Variables *);
int is_only_binary_oper(char );
int is_oper(char );
int maybe_is_correct_sym(char );
double operate(double , char , double );
int priority(char );
int var_index(char );
double var_value(char , const Variables *);
Pair expression_value(const char[], const Variables *, double *);
```

6.7 errors.c

```
#include <stdio.h>
#include <stdlib.h>
#include "mathexpression.h"
#include "errors.h"
```

```
void print_error(Pair pairError) {
    /*Функция, которая выводит информацию об ошибке в поток stderr.
    Принимает структуру Pair с полем number равным номеру ошибки и
   полем index равным индексу места в строке, в которой произошла ошибка.*/

    FILE * file_stream = stderr;
    switch(pairError.number) {
        case CHAR_IS_NOT_ALLOWED_ERROR:
               {
            fprintf(file_stream, "In the line at the character with
            number %d\n", pairError.index);
            fprintf(file_stream, "SyntaxError: the character is not
            allowed\n");
            break;
        }
        case BEFORE_OPERATION_EXPECTED_ERROR:
               {
            fprintf(file_stream, "In the line at the character with
            number %d\n", pairError.index);
            fprintf(file_stream, "OperandNotFoundError: an operand
            expected before the operation\n");
            break;
        }
        case AFTER_OPERAND_EXPECTED_ERROR:
               {
            fprintf(file_stream, "In the line at the character with
            number %d\n", pairError.index);
            fprintf(file_stream, "OperationNotFoundError: an operation
             expected before the operand\n");
            break;
        }
        case NO_ALL_BRACKETS_CLOSED_ERROR:
               {
```

```
        fprintf(file_stream, "In the line at the character with
         number %d\n", pairError.index);
        fprintf(file_stream, "BracketError: not all brackets are
         closed\n");
        break;
    }
    case OPENING_BRACKET_NOT_MET_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "BracketError: an opening bracket was not
        encountered for this closing bracket\n");
        break;
    }
    case BEFORE_POINT_EXPECTED_DIGIT_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "PointError: a digit was expected
        before the dot\n");
        break;
    }
    case AFTER_POINT_EXPECTED_DIGIT_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "PointError: a digit was expected
        after the dot\n");
        break;
    }
    case AFTER_OPERATION_EXPECTED_OPERAND_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
```

```
        number %d\n", pairError.index);
        fprintf(file_stream, "OperandNotFoundError: an operand
        expected after the operation\n");
        break;
    }
    case BRACKETS_IS_EMPTY_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "SyntaxError: brackets are empty\n");
        break;
    }
    case POINT_IS_EXCESS_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "PointError: the point is excess\n");
        break;
    }
    case ZERO_DIVISION_ERROR:
           {
        fprintf(file_stream, "ZeroDivisionError: division by zero\n");
        break;
    }
    case POWER_ERROR:
           {
        fprintf(file_stream, "NotPositiePowerError: raising to a
        non-integer power of a not positive number\n");
        break;
    }
    case EMPTY_STRING_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
```

```
        number %d\n", pairError.index);
        fprintf(file_stream, "SyntaxError: math expression missing\n");
        break;
    }
    case VAR_NOT_DEFINED_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "VarNotDefinedError: the variable was
        not defined\n");
        break;
    }
    case LEFT_PART_NOT_MET_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "SyntaxError: missing left side of
        assignment\n");
        break;
    }
    case COMMAND_NOT_EXIST_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "CommandError: no such command exists\n");
        break;
    }
    case AFTER_COMM_NOT_EXPECTED_CHARS_ERROR:
           {
        fprintf(file_stream, "In the line at the character with
        number %d\n", pairError.index);
        fprintf(file_stream, "SyntaxError: no characters expected
        after this command\n");
```

```
            break;
        }
        case FILE_NOT_FOUND_ERROR:
            {
            fprintf(file_stream, "FileNotFoundError: no file to read
            in this directory was found\n");
            break;
        }
        case MUST_BE_NO_CHARS_BETWEEN_ERROR:
            {
            fprintf(file_stream, "In the line at the character with
            number %d\n", pairError.index);
            fprintf(file_stream, "SyntaxError: there must be no
            characters between the variable and the = symbol\n");
            break;
        }
        case FILE_NAME_NOT_MET_ERROR:
            {
            fprintf(file_stream, "SyntaxError: no file name encountered\n");
            break;
        }
        case FILE_IS_RUNNING_ERROR:
            {
            fprintf(file_stream, "ImportError: trying to run the same file\n")
            break;
        }
    }
}
```

6.8 errors.h

```
#define CHAR_IS_NOT_ALLOWED_ERROR 1
#define BEFORE_OPERATION_EXPECTED_ERROR 2
#define AFTER_OPERAND_EXPECTED_ERROR 3
```

```
#define NO_ALL_BRACKETS_CLOSED_ERROR 4
#define OPENING_BRACKET_NOT_MET_ERROR 5
#define BEFORE_POINT_EXPECTED_DIGIT_ERROR 6
#define AFTER_POINT_EXPECTED_DIGIT_ERROR 7
#define AFTER_OPERATION_EXPECTED_OPERAND_ERROR 8
#define BRACKETS_IS_EMPTY_ERROR 9
#define POINT_IS_EXCESS_ERROR 10
#define ZERO_DIVISION_ERROR 11
#define POWER_ERROR 12
#define EMPTY_STRING_ERROR 13
#define VAR_NOT_DEFINED_ERROR 14
#define LEFT_PART_NOT_MET_ERROR 20
#define COMMAND_NOT_EXIST_ERROR 21
#define AFTER_COMM_NOT_EXPECTED_CHARS_ERROR 22
#define FILE_NOT_FOUND_ERROR 23
#define MUST_BE_NO_CHARS_BETWEEN_ERROR 24
#define FILE_NAME_NOT_MET_ERROR 25
#define FILE_IS_RUNNING_ERROR 26
#define EXIT_NUMBER 30

void print_error(Pair );
```

6.9 stack.c

```
#include "stdlib.h"
#include "stack.h"

Stack* create_stack() {
    /*Функция, которая выделяет на куче память для структуры Stack
    и возвращает указатель на эту область памяти*/

    Stack *tmp = (Stack*) malloc(sizeof(Stack));
    tmp->head = NULL;
```

```
    return tmp;
}


int delete_stack(Stack **stack) {
    /*Функция, которая принимает указатель на указатель на структуру Stack
    и освобождает память, выделенную под нее. В случае успеха
    вовзращает 0, иначе 1.*/

    if (*stack == NULL) {
        return 1;
    }
    Node *tmp = (*stack)->head;
    Node *next = NULL;
    while (tmp) {
        next = tmp->next;
        free(tmp);
        tmp = next;
    }
    free(*stack);
    (*stack) = NULL;
    return 0;
}

int push(Stack *stack, Elem data) {
    /*Функция, которая принимает указатель на структуру Stack и
    значение типа Elem и ставит значение на вершину стека. В случае
    успеха возвращает 0, иначе 1.*/

        Node *tmp = (Node*) malloc(sizeof(Node));
        if (stack == NULL) {
            return 1;
        }
```

```
        tmp->value = data;
        tmp->next = stack->head;

        stack->head = tmp;
        return 0;
}



int pop(Stack *stack, Elem *to_save_here) {
    /*Функция, которая принимает первым аргументом указатель на
    структуру Stack. Если указатель NULL, то возвращает 1, если
    структура пуста, то возвращает 2. В случае успеха возвращает 0,
    а по указателю данному вторым аргументом сохраняется
    значение с вершины стека, а сам элемент с вершины стека удаляется.*/

        Node *prev;
        if (stack == NULL) {
                return 1;
        }

        if (stack->head == NULL) {
                return 2;
        }

        prev = stack->head;
        stack->head = stack->head->next;

        *to_save_here = prev->value;
        free(prev);

        return 0;
}
```

```
int top(Stack *stack, Elem * to_save_here) {
    /*Функция, которая принимает первым аргументом указатель на
    структуру Stack. Если указатель NULL, то возвращает 1, если
    структура пуста, то возвращает 2. В случае успеха возвращает 0,
    а по указателю данному вторым аргументом сохраняется
    значение с вершины стека.*/

    if (stack == NULL) {
                return 1;
        }
        if (stack->head == NULL) {
                return 2;
        }
    *to_save_here = stack->head->value;
    return 0;
}
```

6.10 stack.h

```
typedef union Elem {
    double num;
    char sym;
} Elem;

typedef struct Node {
    Elem value;
    struct Node * next;
} Node;


typedef struct Stack {
    Node *head;
```

```
} Stack;


Stack* create_stack();
int delete_stack(Stack **);
int push(Stack *, Elem);
int pop(Stack *, Elem *);
int top(Stack *, Elem *);
```