**Государственный комитет Российской Федерации по телекоммуникациям**

**Сибирский государственный университет телекоммуникаций и информатики**

**КУРСОВАЯ РАБОТА**

**По дисциплине “Технологии разработки программного обеспечения”**

**Вариант № 4**

Работу выполнил:

                                                                   студент 1 курса

                                                                   Группы:

ПБТ-01

                                                                   Чульдум Саян Андреевич

                                                                   Работу проверил:

### Остапкевич Михаил Борисович

### Работа защищена:

                                                                   «\_\_\_» \_\_\_\_\_\_\_\_\_\_2020г.

                                                                                                                                                                                      С оценкой «\_\_\_\_\_\_\_»

**г. Новосибирск**

**2020**

**СОДЕРЖАНИЕ**

1. Формирование требований
2. Проектирование
   1. Выбор платформы
   2. Выбор языка программирования
   3. Проектирование алгоритма
   4. Модули программы
3. Реализация
   1. Программирование
   2. Документирование – разработка модели
   3. Документирование
4. Оценка производительности

Выводы

Список используемой литературы

1. **ФОРМИРОВАНИЕ ТРЕБОВАНИЙ**

Необходимо разработать шестнадцатеричный калькулятор, который принимает входные значения по имени файла. Входными данными являются выражения с инфиксной формой записи. Программа должна быть совместима с 86-ти разрядной Windows 10.

Программа должна выполнять следующие основные действия:

1. считывать текстовый файл по его имени, файл содержит выражения в шестнадцатеричной системе счисления, файл может содержать несколько выражений, каждая в отдельной строке;
2. открывать исходный файл, обрабатывая возможные ошибочные ситуации;
3. в случае ошибок в выражении, выводить сообщение об ошибке;
4. производит вычисления;
5. печатать результаты вычисления в отдельный файл в шестнадцатеричной системе счисления и завершать работу.

Выражение для калькулятора – это:

* беззнаковая шестнадцатеричная константа;
* (выражение);
* выражение + выражение;
* выражение – выражение;
* выражение \* выражение;
* выражение / выражение;
* not(выражение) – операция побитовой инверсии;
* and(выражение, выражение) – операция побитового И;
* or(выражение, выражение) – операция побитового ИЛИ;
* xor(выражение, выражение) – операция исключающего побитового ИЛИ.

Для задания 4 ось X – размер входного файла со строкой выражения.

1. **ПРОЕКТИРОВАНИЕ**
   1. **Выбор платформы**

Согласно требованиям заказчика, программу необходимо реализовать под операционную систему Windows.

* 1. **Выбор языка программирования**

Для написания кода программы, будет использован язык программирования C, как наиболее подходящий для данной задачи, как и наиболее известный заказчику. Среда программирования – Microsoft Visual Studio Community 2019 version 16.8.3.

* 1. **Проектирование алгоритма**

По запросу заказчика запишем конструкцию выражения для шестнадцатеричного калькулятора по расширенной форме Бэкуса-Наура (рБНФ). При разработке алгоритма будем основываться на данной конструкции:

Выражение = Слагаемое {Сложение Слагаемое}.

Слагаемое = Множитель {Умножение множитель}.

Множитель = Число | Выражение | (Инверсия Выражение) |

Побит\_оп (Выражение, Выражение).

Сложение = “+” | “-”.

Умножение = “\*” | “/”.

Инверсия = not.

Побит\_оп = and | or | xor.

Число = Цифра {Цифра}.

Цифра = “0” | “1” | “2” … |”9” | “A” | “B” … | “F”.

Над самим выражением мы проведем лексический и синтаксический анализы. За лексический анализ отвечает следующая часть конструкции:

Сложение = “+” | “-”.

Умножение = “\*” | “/”.

Инверсия = not.

Побит\_оп = and | or | xor.

Число = Цифра {Цифра}.

Цифра = “0” | “1” | “2” … |”9” | “A” | “B” … | “F”.

Выражение разбивается на лексемы, тип которых определяется исходя из вышеперечисленных требований.

За лексический анализ будет отвечать функция void getToken(void). Лексема, которая была отобрана из выражения записывается в глобальный символьный массив token, а его тип будет записан в char tokType.

Определим три типа лексем:

DELIMITER = Сложение | Умножение | “,” | “(” | “)”.

IDENTIFIER = Инверсия | Побит\_оп.

NUMBER = Число.

Для удобства лексического анализа создадим функцию void getExpr(FILE \*fp, char \*pStr), которая убирает из прочтенной строки пробельные символы и заменяет буквы на прописные.

За синтаксический анализ отвечает остальная часть рБНФ конструкции:

Выражение = Слагаемое {Сложение Слагаемое}.

Слагаемое = Множитель {Умножение множитель}.

Множитель = Число | Выражение | (Инверсия Выражение) |

Побит\_оп (Выражение, Выражение).

Синтаксический анализ будет основан на методе рекурсивного спуска. За каждый терминал, стоящий слева от “=” будет отвечать отдельная функция.

Код лексического и синтаксического анализа будет реализован в отдельном модуле analysis.h

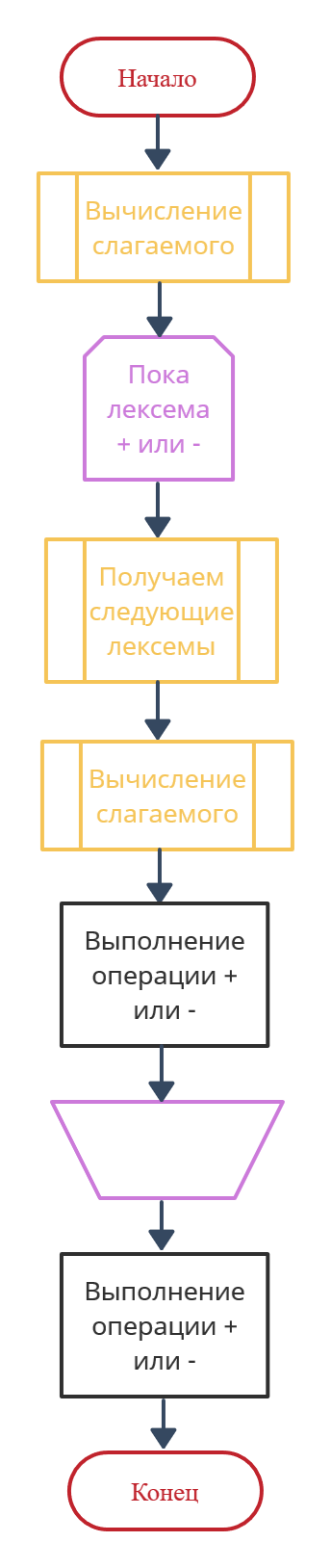


Рис. 1. Блок-схема алгоритма функции, возвращающей значение выражения.

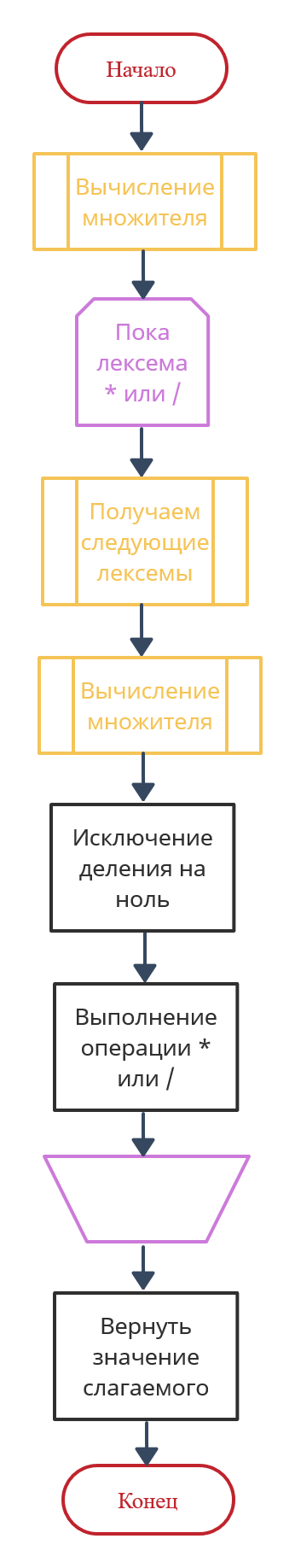
****

Рис. 2. Блок-схема алгоритма функции, возвращающей значение слагаемого.

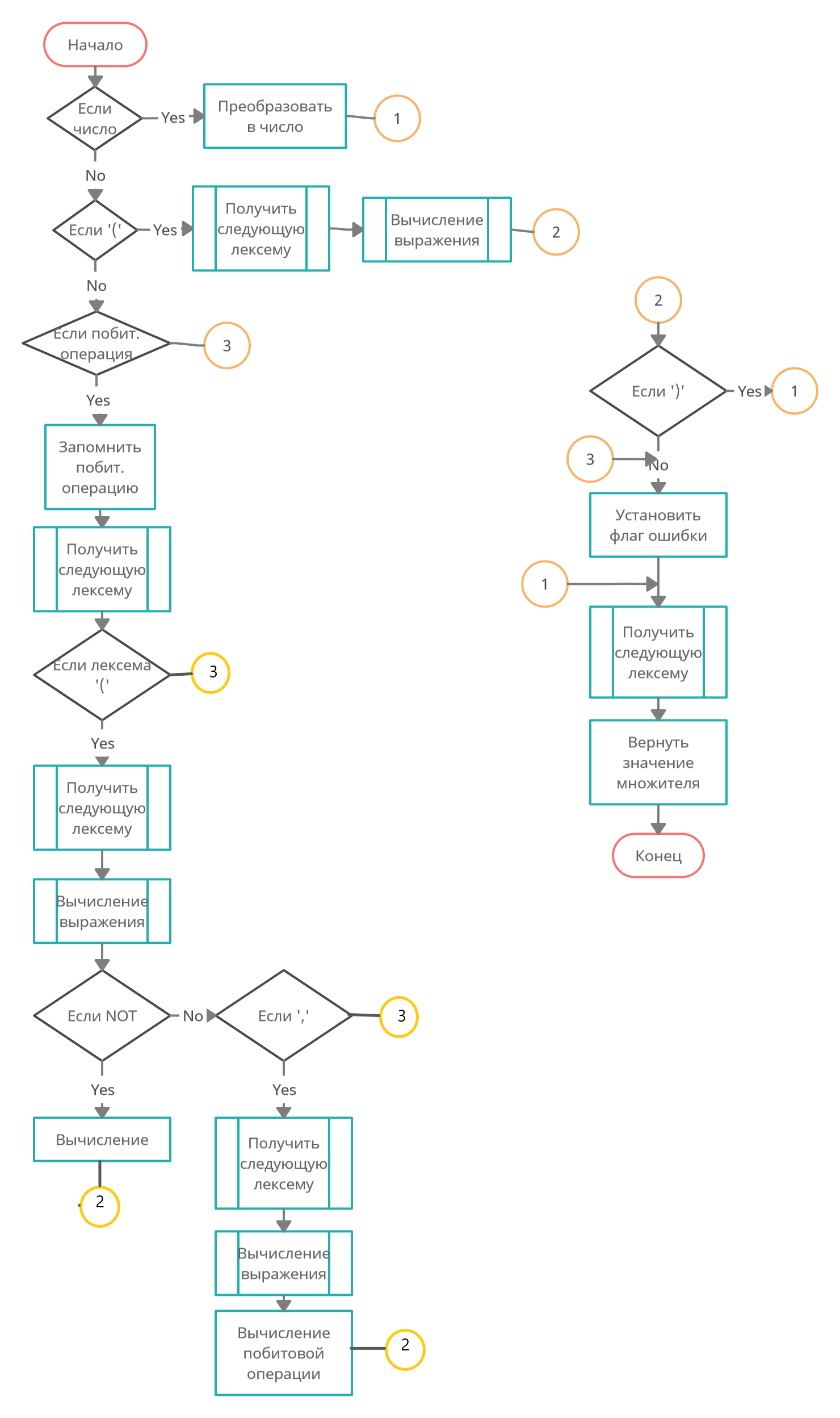


Рис. 3. Блок-схема алгоритма функции, возвращающей значение множителя.

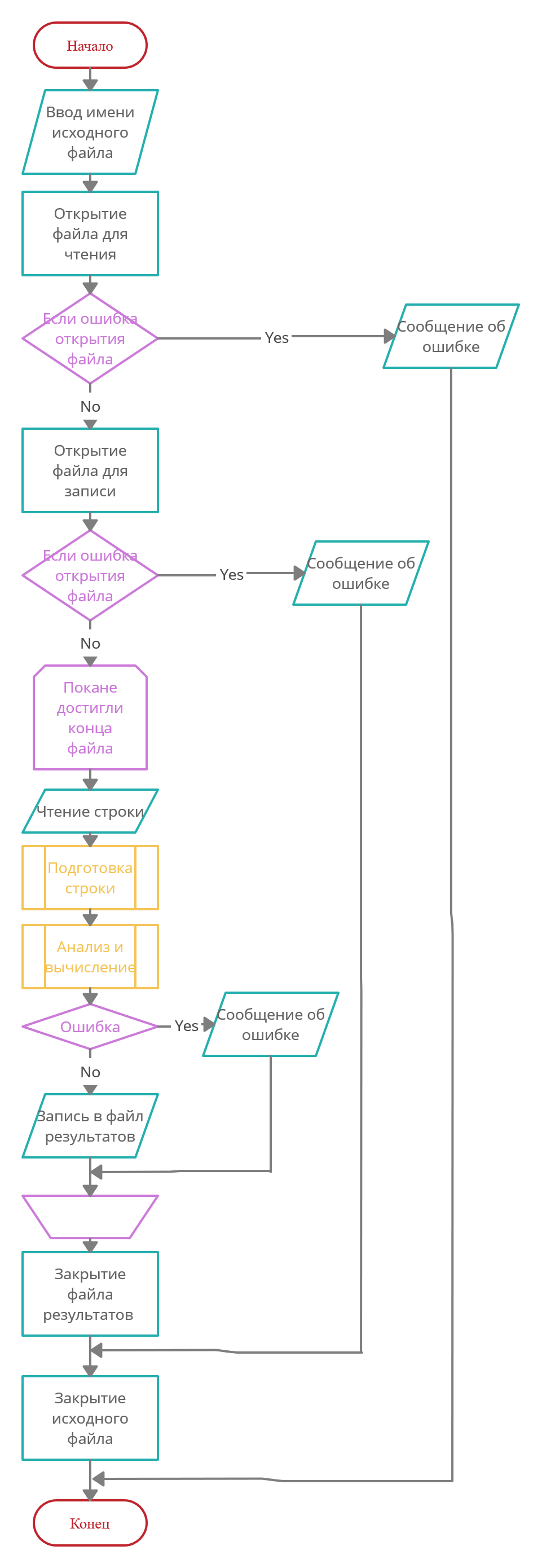


Рис. 4. Блок-схема алгоритма основной программы

* 1. **Модули программы**

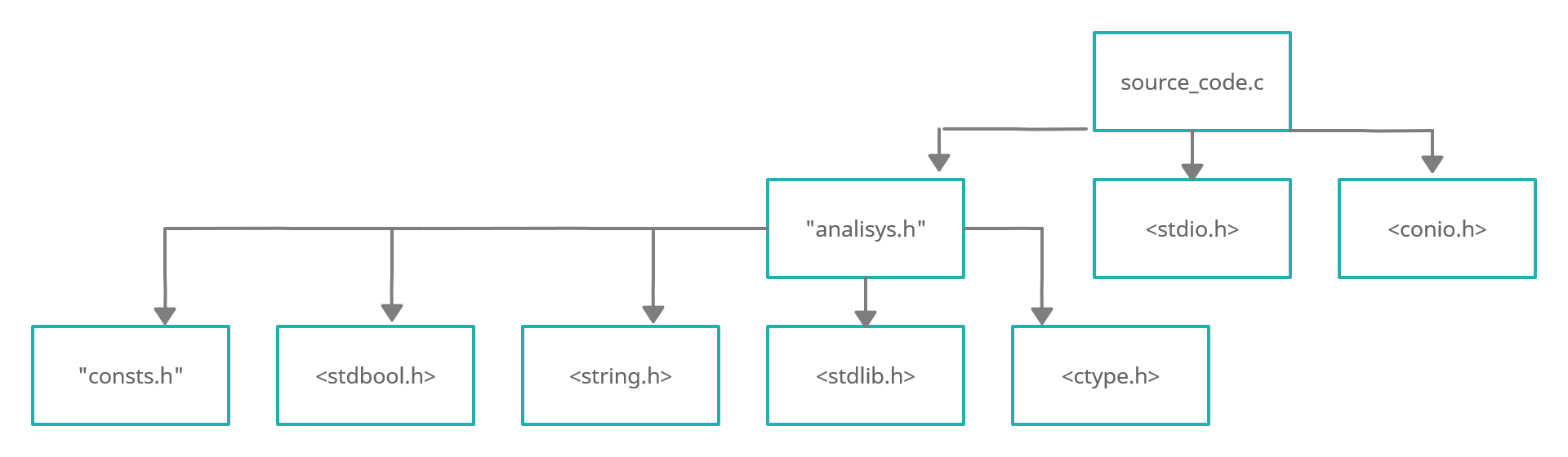


Рис. 5. Модули основной программы

1. **РЕАЛИЗАЦИЯ**
   1. **Программирование**

Исходный текст модуля consts.h

/\*

\* Модуль значений констант для анализатора

\*/

/\*Error codes\*/

#define UNKNOWN\_SYMB 0

#define SYNTAX\_ERR 1

#define OPEN\_BR\_EXP 2

#define CLOSE\_BR\_EXP 3

#define DELIMITER\_EXP 4

#define DIV\_BY\_ZERO 5

#define EXCESS\_OF\_THE\_MAX 6

/\*Token types\*/

#define UNKNOWN 7

#define DELIMITER 8

#define NUMBER 9

#define IDENTIFIER 10

/\*Buffer lengths\*/

#define GREAT\_S 255

#define SMALL\_S 11

Исходный текст модуля analysis.h

/\*

\* Модуль выполняет лексический и синтаксический анализ выражения

\* в шестнадцатеричной системе исчисления, вида:

\*

\* • беззнаковая шестнадцатеричная константа;

\* • (выражение)

\* • выражение + выражение

\* • выражение - выражение

\* • выражение \* выражение

\* • выражение / выражение

\* • not(выражение) – операция побитовой инверсии

\* • and(выражение, выражение) – операция побитового И

\* • or(выражение, выражение) – операция побитового ИЛИ

\* • xor(выражение, выражение) – операция исключающего побитового ИЛИ

\*

\* на которое указывает pExpr и производит вычисление.

\*

\* Автор Чульдум Саян Андреевич, 2021г.

\*/

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <stdbool.h>

#include "consts.h" //модуль постоянных величин

char \*pExpr; // указатель на анализируемое выражение

char token[GREAT\_S]; // буфер для выделенной лексемы

char tokenType; // тип выделенной лексемы

char errorType; // тип ошибки

bool error; // флаг об ошибке в выражении

bool calculate(char \*pStr, unsigned long \*Result);

unsigned long expression(void);

unsigned long summation(void);

unsigned long multiplier(void);

void getToken(void);

void seterror(char Type);

char \*errorMessage(void);

/\*

Точка входа анализатора.

\*/

bool calculate(char \*pStr, unsigned long \*result) {

error = false; // сбрасываем флаг ошибки

pExpr = pStr; // адрес выражения передаем глобальному указателю

getToken(); // получаем первую лексему

\*result = expression(); // определяем значение выражения

if(\*pExpr && !error) seterror(SYNTAX\_ERR); // если после вычисления указатель не в конце строки, значит произошла ошибка

return !error;

}

/\*

Функция возвращает значение выражения

\*/

unsigned long expression(void) {

unsigned long result = 1;

char operation;

if (!error) { // если небыло ошибки

result = summation(); // находим значение первого слагаемого

while ((operation = \*token) == '+' || operation == '-'){

getToken();

switch(operation) {

case '-':

result = result - summation();

break;

case '+':

result = result + summation();

break;

}

}

}

return result;

}

/\*

Функция возвращает значение слагаемого

\*/

unsigned long summation(void) {

unsigned long result = 1, temp;

char operation;

if (!error) {

result = multiplier(); // получаем значение первого множителя

while ((operation = \*token) == '\*' || operation == '/'){

getToken();

switch(operation) {

case '\*':

result = result \* multiplier();

break;

case '/':

temp = multiplier();

if (temp == 0) seterror(DIV\_BY\_ZERO); // исключение деления на 0

else result = result / temp;

break;

}

}

}

return result;

}

/\*

Функция возвращает значение множителя

\*/

unsigned long multiplier(void) {

unsigned long result = 1;

char operation;

if (!error){

if (tokenType == NUMBER) sscanf(token, "%x", &result); // если тип полученной лексемы - число, получаем его значение

else if (\*token == '(') { // если полученная лексема - открывающая скобка

getToken();

result = expression(); // получаем значение выражения в скобках

if (!error && \*token != ')') seterror(CLOSE\_BR\_EXP); // ожидаем закрывающую скобку после выражения

}

else if (tokenType == IDENTIFIER){ // если тип лексемы - идентификатор функции

operation = \*token; // запоминаем первый символ побитовой операции

getToken();

if (\*token == '(') { // после лексемы побитовой операции ожидаем скобку

getToken();

result = expression(); // получаем результат первого аргумента

if (operation == 'N') result = ~result; // выполнение операции побитовой инверсии

else { // если операция не NOT получаем второй аргумент

if (\*token == ',') { // ожидаем разделитель аргументов

getToken();

switch(operation){

case 'A': // побитовое И

result = result & expression();

break;

case 'O': // побитовое ИЛИ

result = result | expression();

break;

case 'X': // побитовое исключающее ИЛИ

result = result ^ expression();

break;

}

}

else if (!error) seterror(DELIMITER\_EXP); // если не обнаружили запятую после первого аргумента

}

if (!error && \*token != ')') seterror(CLOSE\_BR\_EXP); // ожидаем закрывающую скобку после аргументов функции

}

else seterror(OPEN\_BR\_EXP); // если не обнаружили открывающую скобку

}

else seterror(SYNTAX\_ERR); // если лексема или ее тип не определена

getToken();

}

return result;

}

/\*

Функция возвращает очередную лексему

\*/

void getToken(void) {

char \*pChar;

tokenType = UNKNOWN;

pChar = token;

\*pChar = '\0'; //нуль-терминирование символьной строки

if(!\*pExpr || error) return; // конец выражения или ошибка

/\* выделение знаков пунктуации и операторов\*/

if(strchr("+-\*/(),", \*pExpr)){

tokenType = DELIMITER;

\*pChar++ = \*pExpr++; // переходим к следующему символу

}

/\*выделение побитовых операций\*/

else if (memcmp(pExpr, "OR", 2) == 0){

for (int i = 0; i < 2; i++) \*pChar++ = \*pExpr++;

tokenType = IDENTIFIER;

}

else if (memcmp(pExpr, "AND", 3) == 0 ||

memcmp(pExpr, "NOT", 3) == 0 ||

memcmp(pExpr, "XOR", 3) == 0){

for (int i = 0; i < 3; i++) \*pChar++ = \*pExpr++;

tokenType = IDENTIFIER;

}

/\*выделение шестнадцатеричных чисел\*/

else if(isxdigit(\*pExpr)){

while(isxdigit(\*pExpr)) \*pChar++ = \*pExpr++;

if (strlen(token) > 8) seterror(EXCESS\_OF\_THE\_MAX); // число превышает макс. доп. значение

else tokenType = NUMBER;

}

/\*ошибочный символ в выражении\*/

else seterror(UNKNOWN\_SYMB);

\*pChar = '\0'; //нуль-терминирование символьной строки

}

/\*

Функция устанавливает флаг и тип ошибки

\*/

void seterror(char Type) {

error = true;

errorType = Type;

}

/\*

Функция возвращает указатель на строку с сообщением об ошибке

\*/

char \*errorMessage(void) {

static char \*errorStr[]= {

"Недопустимый символ в выражении",

"Синтаксическая ошибка в выражении",

"Ожидается '('",

"Ожидается ')'",

"Ожидается ','",

"Деление на ноль!",

"Превышение максимального значения числа"

};

return errorStr[errorType];

}

Исходный текст модуля основной программы

/\*

\* Программа шестнадцатеричный калькулятор с инфиксной формой записи

\* считывает и вычисляет выражения из входного текстового файла, вида:

\* • беззнаковая шестнадцатеричная константа;

\* • (выражение)

\* • выражение + выражение

\* • выражение - выражение

\* • выражение \* выражение

\* • выражение / выражение

\* • not(выражение) – операция побитовой инверсии

\* • and(выражение, выражение) – операция побитового И

\* • or(выражение, выражение) – операция побитового ИЛИ

\* • xor(выражение, выражение) – операция исключающего побитового ИЛИ

\*

\* Результат сохраняет в новом текстовом файле с окончанием "\_res.txt".

\*

\* Автор Чульдум Саян Андреевич, 2021г.

\*/

#include <stdio.h>

#include <conio.h>

#include "analysis.h" //подключаем модуль который выполняет синтаксический и лексический анализы

void getExpr(FILE \*fp, char \*pStr);

int main(void) {

FILE \*sourceFile, \*resultFile;

char fileName[GREAT\_S], expr[GREAT\_S];

unsigned long result;

//открываем исходный файл

printf("Enter the file name: ");

scanf("%s", fileName);

if ((sourceFile = fopen(fileName,"r")) == NULL) {

printf("ERROR. Cannot open the file.");

getchar();

return 0;

}

strcat(fileName, "\_res.txt"); // имя файла результата

//открываем файл для записи результата

if ((resultFile = fopen(fileName,"w")) == NULL) {

printf("ERROR. Cannot open/create the result file.");

getchar();

fclose(sourceFile);

return 0;

}

while (!feof(sourceFile)) {

getExpr(sourceFile, expr); // получаем строку с выражением

if (!\*expr) continue; // если строка пустая - не выполняем никаких действий

fprintf(resultFile, "%s = ", expr); // печатаем строку с выражением в файл

if (calculate(expr, &result)) fprintf(resultFile, "%X\n", result); // если вычисление прошло без ошибок записываем ответ

else fprintf(resultFile, "ERROR! %s\n", errorMessage()); // иначе записываем сообщение об ошибке

}

fclose(sourceFile);

fclose(resultFile);

printf("The calculation results are written to file %s", fileName);

getchar();

return 1;

}

/\*

Функция пропускает лишние пробелы,

переводит буквы в верхний регистр

и присваивает видоизмененную строку

переменной expr в main

\*/

void getExpr(FILE \*fp, char \*pStr) {

int ch; //текущий символ

char \*pChar;

pChar = pStr;

while ((ch = getc(fp)) != '\n' && ch != EOF) { // читаем символы пока не встретим перевод строки, или конец файла

if (isspace(ch)) continue; // пропускаем пробельные символы

\*pChar = toupper(ch); // переводим буквенные символы в верхний регистр

pChar++;

}

\*pChar = '\0'; //нуль-терминирование символьной строки

}

* 1. **Документирование – разработка модели**

Ниже представлена версия модели документации для руководства пользователя. Описаны лишь основные моменты работы с просто программой, как нашей.

Оглавление (общий раздел) ( ).

Раздел А: Введение ( ).

I. Назначение ( ).

II. Выполняемые функции ( ).

III. Ограничения ( ).

Раздел В: Общая информация ( ).

I. Допустимые элементы входных данных ( ).

II. Выходные данные ( ).

Раздел С: Выполнение задания ( ).

II. Загрузка/запуск программы (название диалоговой программы) ( ).

III. Ввод входных данных ( ).

IV. Процедура обработки ошибок ( ).

V. Изменение входных данных ( ).

VI. Получение выходных данных ()

VII. Завершение работы (название диалоговой программы) ( ).

* 1. **Документирование**

***Руководство пользователя программы “Шестнадцатеричный калькулятор”***

***Раздел А: Введение***

1. ***Назначение***

Программа считывает шестнадцатеричные выражения, записанные в исходном текстовом файле, выполняет необходимые вычисления и записывает их результат в отдельный, новый файл в шестнадцатеричной системе счисления.

1. ***Выполняемые функции.***

Программа запрашивает имя исходного файла с заранее записанными выражениями. Поочередно считывает строки, выполняя вычисления и записывая результат в результирующий файл. По окончании работы со строкой, программа переходит на следующую строку и так до конца файла. В случае обнаружения ошибки чтения/записи файла, а также ошибки в записи выражений, выдает ошибку на консоль или же записывает ошибку вычисления непосредственно в результирующий файл. После обработки, программа завершает работу.

1. ***Ограничения.***

Длина строки не должна превышать 255 символов (кроме пробельных символов). Верхняя граница допустимых значений – FFFFFFFF (4 байта). Программа вычисляет выражения следующего типа:

* + - беззнаковая шестнадцатеричная константа;
    - (выражение)
    - выражение + выражение
    - выражение - выражение
    - выражение \* выражение
    - выражение / выражение
    - not(выражение) – операция побитовой инверсии
    - and(выражение, выражение) – операция побитового И
    - or(выражение, выражение) – операция побитового ИЛИ
    - xor(выражение, выражение) – операция исключающего побитового ИЛИ

***Раздел B: Общая информация***

1. ***Допустимые элементы входных данных.***

Каждое выражение записывается в отдельной строке. Знак ‘=’ в конце не ставится. Количество выражений неограниченно.

1. ***Выходные данные.***

Результаты вычислений записываются в новом файле *“<имя исходного файла>\_res.txt”* в формате:

<выражение> = <результат>

В случае ошибки:

<выражение> = <сообщение об ошибке>

***Раздел C: Выполнение задания***

1. ***Загрузка/запуск программы “Шестнадцатеричный калькулятор”***

Для запуска приложения используется исполняемый файл, после запуска которого программа приглашает пользователя к вводу имени файла.

1. ***Ввод входных данных.***

Для работы программы необходимо ввести имя исходного файла с записанными выражениями. Возможно записать путь к файлу в абсолютном виде – c:/Users/User/Desktop/*имя\_файла.txt*, а также в относительном – ../*имя\_файла.txt.* В случае верного ввода программа начнет вычисления. Если исходный файл открыть не удалось, то программа завершит работу с ошибкой.

1. ***Процедура обработки ошибок.***

Программа способна обнаруживать и обрабатывать следующие ошибки в выражениях:

• ошибки в лексемах;

• превышение максимального значение числа;

• синтаксические ошибки;

• ошибки расстановки скобок и разделителей;

• деление на ноль.

При обнаружении какой-либо ошибки, программа записывает текст ошибки в результирующем файле в формате:

<выражение> = <сообщение об ошибке>.

1. ***Изменение входных данных.***

Исходный файл изменяется пользователем, посредством открытия файла любым текстовым редактором. Перед вычислением новых значений, необходимо сохранить или переименовать результирующий файл со старыми значениями.

1. ***Получение входных данных.***

Чтобы ознакомиться с результатами вычислений, необходимо открыть файл *имя\_файла\_res.txt* любым текстовым редактором.

1. ***Завершение работы программы “Шестнадцатеричный калькулятор”.***

Программа завершает свою работу при выводе соответствующего сообщения, либо при выводе ошибки.

1. **ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ**

Протестируем программу на различных уровнях оптимизации.

* Od – оптимизация отключена;
* O1 – наименьший размер;
* O2 – наибольшая скорость;
* Ox – Полная оптимизация;

Для этого подключим к программе модуль *benchmark.h.*

Составим таблицу с результатами измерения времени выполнения программы, а также на основе этих данных построим график зависимости времени выполнения программы от уровня оптимизации и от размера исходного файла.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Уровень оптимизации | Размер входного файла, кбайт | | | | | | | | | |
| 1,31 | 2,60 | 3,86 | 5,17 | 6,47 | 7,82 | 9,00 | 10,30 | 11,60 | 13,00 |
| O1 | 1,642 | 2,169 | 2,802 | 3,247 | 3,766 | 4,047 | 4,747 | 5,250 | 5,806 | 6,639 |
| O2 | 1,501 | 2,036 | 2,507 | 3,087 | 3,573 | 4,005 | 4,531 | 4,881 | 5,416 | 6,117 |
| Od | 1,619 | 2,326 | 2,761 | 3,221 | 3,838 | 4,280 | 4,884 | 5,292 | 5,969 | 6,509 |
| Ox | 1,588 | 2,049 | 2,511 | 3,120 | 3,650 | 4,113 | 4,588 | 5,005 | 5,545 | 6,087 |

Из графика видно, что время выполнения программы прямо пропорционально размеру исходного файла, в то же время уровень оптимизации практически ничего не решает, хотя возможно при увеличении количества данных, разница будет значительной.