

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
«Московский государственный технический университет имени Н.Э. Баумана»  
(МГТУ им. Н.Э. Баумана)



Факультет Информатики и системы управления

Кафедра Программное обеспечение ЭВМ и информационные технологии

**Отчет по практике**

Студент Жарова Екатерина Александровна  
(фамилия, имя, отчество)

Группа ИУ7-21

Название практики \_\_\_\_\_

Руководитель

\_\_\_\_\_  
Преподаватель  
Должность

\_\_\_\_\_  
ФИО

\_\_\_\_\_  
подпись

1. Установка программного обеспечения .....	3
2. Этапы компиляции программы .....	4
3. Исследование исполняемого файла .....	7
4. Работа с QT Creator .....	11
5. Индивидуальное задание.....	14
6. Исследование покрытия кода тестами .....	17
Заключение .....	19

# 1. Установка программного обеспечения

*ОС, на которой выполнялась установка ПО: Windows 7 64-bit*

*Основные шаги установки:*

1. По предложенной ссылке был скачан и установлен на компьютер Qt Creator 3.6.0 (Проблем с установкой не возникло). Установка выполнялась согласно краткому предписанию из методических указаний:
  - 1) загрузите программу-инсталлятор
  - 2) запустите её;
  - 3) оставьте все настройки по умолчанию;
  - 4) согласитесь с лицензией;
  - 5) нажмите на кнопку «Установить».
2. По предложенной ссылке была скачана нужная версия MinGW. Была запущена установка, в ходе которой были выбраны нужные компоненты. Далее была произведена загрузка необходимых пакетов.
3. Перед запуском первой программы была произведена настройка Qt Creator 3.6.0 в соответствии с инструкцией, описанной в методических указаниях.

## 2. Этапы компиляции программы

При помощи следующей программы изучались *этапы компиляции* программы:  
(Программа считывает все числа до ввода символа и суммирует их по модулю)

```
#include <stdio.h>
#define ABS(v) (v) < 0 ? -(v) : (v)

int main(void)
{
    int pval, ABS_sum;
    ABS_sum = 0;
    printf("Input numbers:\n");
    while (scanf("%d", &pval) > 0) //проверка на корректность
    {
        ABS_sum += ABS(pval);
    }
    printf("ABS sum = %d", ABS_sum);

    return 0;
}
```

Из презентации L\_01 нам известны стадии компиляции:

1. Обработка препроцессором
2. Трансляция на язык ассемблера
3. Ассемблирование в объектный файл
4. Компоновка

Рассмотрим каждую из них подробнее:

1. *Обработка препроцессором*

*Команда:* **cpp -o t01.i t01.c**

*Результат работы:* создание t01.i

t01.c - 342 байт, t01.i - 18 795 байт

```
...
extern int __attribute__((__cdecl__)) __attribute__((__nothrow__)) __mingw_printf(const
char*, ...);
...
int main(void)
{
    int pval, ABS_sum;

    ABS_sum = 0;
    printf("Input numbers:\n");
    while (scanf("%d", &pval) > 0)
    {
        ABS_sum += (pval) < 0 ? -(pval) : (pval);
    }
    printf("ABS sum = %d\n", ABS_sum);

    return 0;
}
...
```

## 2. Трансляция на язык ассемблера

Команда: **C:\MinGW\bin\c99 -S -masm=intel t01.i**

(Изначально использовалась команда: `c99 -S -masm=intel t01.i`, но это приводило к ошибке: "Not such file or directory")

Результат работы: создание t01.s

t01.c - 342 байт, t01.s - 7 344 байт

Функция main:

\_main:

LFB10:

```
.cfi_startproc
push    ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
mov     ebp, esp
.cfi_def_cfa_register 5
and     esp, -16
sub     esp, 32
call    __main
mov     DWORD PTR [esp+28], 0
mov     DWORD PTR [esp], OFFSET FLAT:LC0
call    _puts
jmp     L32
```

## 3. Ассемблирование в объектный файл

Команда: **as -o t01.o t01.s**

Результат работы: создание t01.o

t01.c - 342 байт, t01.o - 13 346 байт

## 4. Компоновка

Команда: **C:\MinGW\bin\gcc t01.o -o t01**

Результат работы: создание t01.exe

t01.c - 342 байт, t01.exe - 83 122 байт

Можно получить t01.exe при помощи ld, используя команду: **gcc -v -o t01.exe t01.o:**

```
ld c:/mingw/bin/./libexec/gcc/mingw32/4.9.3/collect2.exe -plugin
c:/mingw/bin/./libexec/gcc/mingw32/4.9.3/liblto_plugin-0.dll -plugin-
opt=c:/mingw/bin/./libexec/gcc/mingw32/4.9.3/lto-wrapper.exe -plugin-opt=-
fresolution=C:\Users\caterina\AppData\Local\Temp\ccjEAE0e.res -plugin-opt=-pass-
through=-lmingw32 -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_eh -
plugin-opt=-pass-through=-lmoldname -plugin-opt=-pass-through=-lmingwex -plugin-opt=-
pass-through=-lmsvcrt -plugin-opt=-pass-through=-ladvapi32 -plugin-opt=-pass-through=-
lshell32 -plugin-opt=-pass-through=-luser32 -plugin-opt=-pass-through=-lkernel32 -plugin-
opt=-pass-through=-lmingw32 -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-
through=-lgcc_eh -plugin-opt=-pass-through=-lmoldname -plugin-opt=-pass-through=-
lmingwex -plugin-opt=-pass-through=-lmsvcrt -Bdynamic -o t01.exe
c:/mingw/bin/./lib/gcc/mingw32/4.9.3/./././crt2.o
c:/mingw/bin/./lib/gcc/mingw32/4.9.3/crtbegin.o -Lc:/mingw/bin/./lib/gcc/mingw32/4.9.3 -
Lc:/mingw/bin/./lib/gcc -Lc:/mingw/bin/./lib/gcc/mingw32/4.9.3/././././mingw32/lib -
Lc:/mingw/bin/./lib/gcc/mingw32/4.9.3/././.. t01.o -lmingw32 -lgcc -lgcc_eh -lmoldname -
lmingwex -lmsvcrt -ladvapi32 -lshell32 -luser32 -lkernel32 -lmingw32 -lgcc -lgcc_eh -
lmoldname -lmingwex -lmsvcrt c:/mingw/bin/./lib/gcc/mingw32/4.9.3/crtend.o
```

**Сборка при помощи утилиты Makefile:**

Утилиты не было на компьютере и поэтому она была скачана из Интернета и добавлена в корень "C:\". Makefile для данной задачи выглядит следующим образом:

```
FLAGS :=-std=c99 -Wall -Werror -pedantic
```

```
ifeq ($(mode), debug)
```

```
    FLAGS += -g3
```

```
endif
```

```
t01.exe: t01.o
```

```
    gcc $^ -o $@
```

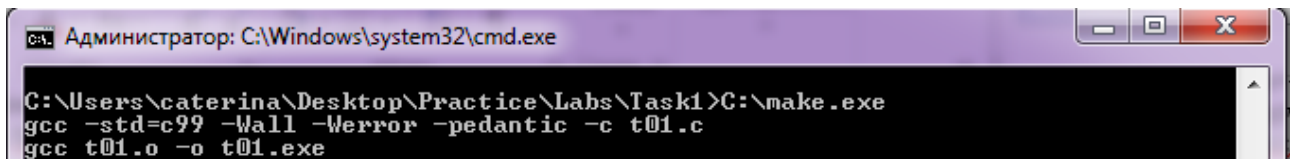
```
t01.o: t01.c
```

```
    gcc $(FLAGS) -c $<
```

```
clean:
```

```
    $(RM) *.o *.exe
```

Результат работы :



```
Администратор: C:\Windows\system32\cmd.exe

C:\Users\caterina\Desktop\Practice\Labs\Task1>C:\make.exe
gcc -std=c99 -Wall -Werror -pedantic -c t01.c
gcc t01.o -o t01.exe
```

### 3. Исследование исполняемого файла

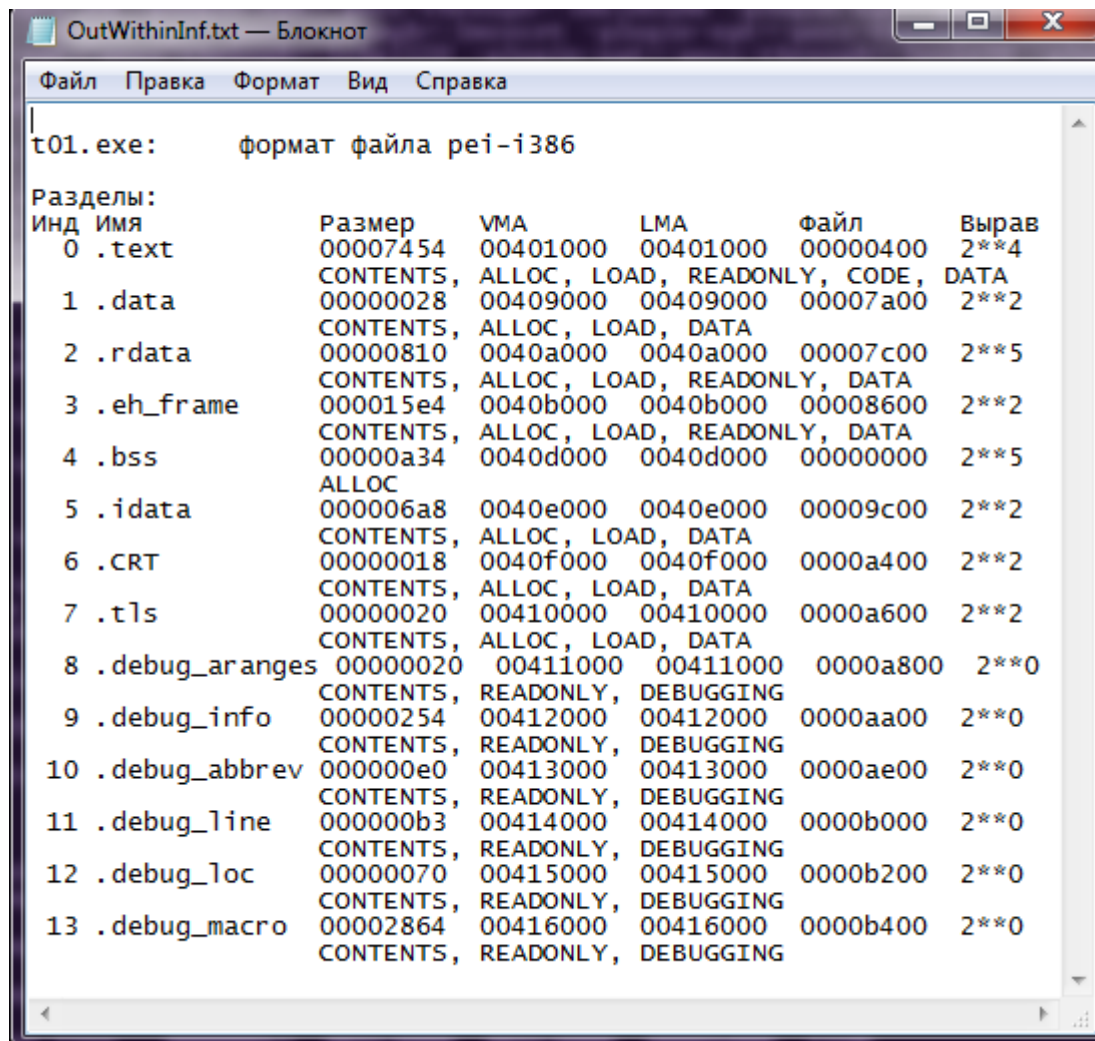
**Сборка с отладочной информацией:**

**Команда:** `c99 -o t01.exe -pedantic -Wall -Werror -g3 -gdwarf-2 t01.c`

(Изначально использовалась команда: `c99 t01.exe -pedantic -Wall -Werror -g3 -gdwarf-2 t01.c`, что приводило к следующему: “ошибка: выполнение ld завершилось с кодом возврата 1”)

**Размер исполняемого файла:** 83 122 байт

Чтобы определить, какие секции входят в программу, нужно воспользоваться следующей командой: `objdump -h t01.exe >OutWithInf.txt`. В созданном файле мы увидим следующее:

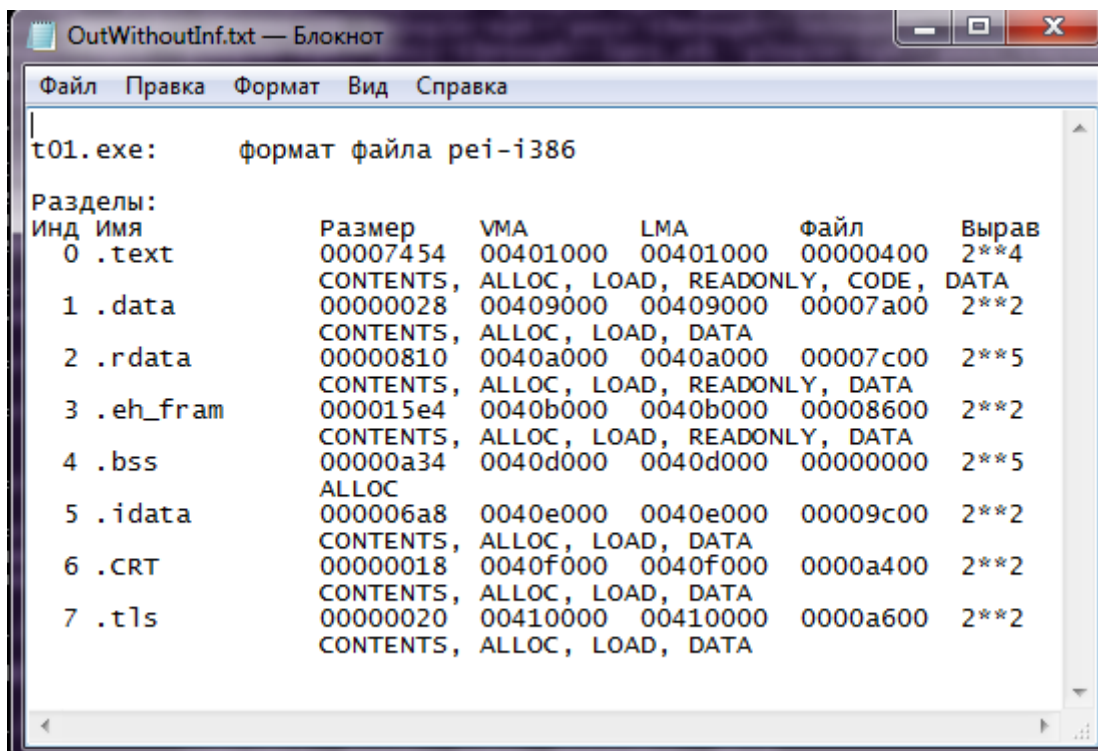


**Сборка без отладочной информации:**

**Команда:** `t01.exe -pedantic -Wall -Werror t01.c`

**Размер исполняемого файла:** 68 918 байт

Чтобы определить, какие секции входят в программу, нужно воспользоваться следующей командой: `objdump -h t01.exe >OutWithoutInf.txt`. В созданном файле мы увидим следующее:



Размер исполняемого файла с отладочной информацией больше на 14 204 байт размера файла без отладочной информации.

**Определение в какие секции попадают переменные (глобальные и локальные) и функции:**

Скомпилируем файл, зафиксируем необходимую информацию:

**c99 -Wall -Werror -pedantic -o t02.exe t01.c**

**objdump -h t02.exe >OutInf1.txt**

Добавим в скомпилированный файл одну. Перекомпилируем файл, зафиксируем изменения:

**c99 -Wall -Werror -pedantic -o t02.exe t01.c**

**objdump -h t02.exe >OutInf2.txt**

Сравним файлы, полученные при помощи утилиты objdump сразу после первой и второй компиляции:

**fc OutInf1.txt OutInf2.txt >ResOutInf.txt**

Откроем ResOutInf.txt и увидим следующее:

Сравнение файлов OutInf1.txt и OUTINF2.TXT

\*\*\*\*\* OutInf1.txt

Инд	Имя	Размер	VMA	LMA	Файл	Вырав
0	.text	00007454	00401000	00401000	00000400	2**4
	CONTENTS, ALLOC, LOAD, READONLY, CODE, DATA					

\*\*\*\*\* OUTINF2.TXT

Инд	Имя	Размер	VMA	LMA	Файл	Вырав
0	.text	00007464	00401000	00401000	00000400	2**4
	CONTENTS, ALLOC, LOAD, READONLY, CODE, DATA					

\*\*\*\*\*

Добавим глобальную переменную int tmp = 1. Посмотрим, куда она попадет:

**c99 -Wall -Werror -pedantic -o t02.exe t01.c**

**objdump -x t02.exe >OTmpInf.txt**



```
[1027](sec 5)(fl 0x00)(ty 0)(scl 2)(nx 0) 0x00000a24 _tmp
Так же найдем функцию main:
[112](sec 1)(fl 0x00)(ty 20)(scl 2)(nx 0) 0x00000930 ____main
```

Вывод: функции попадают в раздел “.text”, а переменные в “data”.

*Глобальный неинициализированный массив:*

Скомпилируем файл, зафиксируем необходимую информацию:

```
c99 -o t02.exe -Wall -Werror -pedantic t01.c
objdump -h t02.exe >OutNMas1.txt
```

Добавим глобальный неинициализированный массив `int mas[10]`; Перекомпилируем файл, зафиксируем изменения:

```
c99 -o t02.exe -Wall -Werror -pedantic t01.c
objdump -h t02.exe >OutNMas2.txt
```

Сравним файлы:

```
fc OutNMas1.txt OutNMas2.txt
```

Сравнение файлов OutNMas1.txt и OUTNMAS2.TXT

```
***** OutNMas1.txt
      CONTENTS, ALLOC, LOAD, READONLY, DATA
4 .bss      00000a34 0040d000 0040d000 00000000 2**5
      ALLOC
***** OUTNMAS2.TXT
      CONTENTS, ALLOC, LOAD, READONLY, DATA
4 .bss      00000a78 0040d000 0040d000 00000000 2**5
      ALLOC
*****
```

Глобальный неинициализированный массив попал в секцию .bss. Размер исполняемого файла изменился с 68 918 байт до 68 936 байт при добавлении массива.

**Глобальный инициализированный массив:**

Скомпилируем файл, зафиксируем необходимую информацию:

```
c99 -o t02.exe -Wall -Werror -pedantic t01.c
objdump -h t02.exe >OutInMas1.txt
```

Добавим глобальный инициализированный массив `int mas[10] = {1,-2,3,-4,5,-6,7,0,9,-9}`; Перекомпилируем файл, зафиксируем изменения:

```
c99 -o t02.exe -Wall -Werror -pedantic t01.c
objdump -h t02.exe >OutInMas2.txt
```

Сравним файлы:

```
fc OutInMas1.txt OutInMas2.txt
```

Сравнение файлов OutInMas1.txt и OUTINMAS2.TXT

```
***** OutInMas1.txt
      CONTENTS, ALLOC, LOAD, READONLY, CODE, DATA
1 .data     00000028 00409000 00409000 00007a00 2**2
      CONTENTS, ALLOC, LOAD, DATA
***** OUTINMAS2.TXT
      CONTENTS, ALLOC, LOAD, READONLY, CODE, DATA
1 .data     00000068 00409000 00409000 00007a00 2**5
```

## CONTENTS, ALLOC, LOAD, DATA

\*\*\*\*\*

Глобальный инициализированный массив попал в секцию .data. Размер исполняемого файла изменился с 68 918 байт до 68 936 байт при добавлении массива.

**Используемые библиотеки:**

**objdump -p t02.exe**

...

DLL Name: KERNEL32.dll

...

msvcrt.dll

## 4. Работа с QT Creator

Копируем исходный код программы из Приложения А в отдельный файл и создаем в QT Creator проект для этой программы.

### Компиляция и сборка.

*Сборка и запуск:* добавляем компилятор MinGW и отладчик gdb-i686-pc-mingw32.exe. Добавляем комплект с добавленными отладчиком и компилятором и называем его MinGW. Удаляем автоматически созданные этапы сборки. Добавляем 2 основных этапа сборки:

1. **gcc -Wall -Werror -pedantic -c pr\_a.c**
2. **gcc -Wall -Werror -pedantic -o pr\_a.exe pr\_a.o**

На вкладке «Запуск» в поле «Программа» пишем pr\_a.exe и ставим галочку у пункта «Запускать в терминале».

*Отладка программы:*

### Синтаксические ошибки:

1. *Ошибка:*  
pr\_a.c:1:19: фатальная ошибка: stdio.h: No such file or directory  
*Исправление:*  
#include <stdio.h>
2. *Ошибка:*  
pr\_a.c: В функции «main»:  
pr\_a.c:5:5: ошибка: format «%f» expects argument of type «float \*», but argument 2 has type «int \*» [-Werror=format=]  
*Исправление:*  
scanf("%d", &max);
3. *Ошибка:*  
pr\_a.c: В функции «main»:  
pr\_a.c:6:24: ошибка: «num» undeclared (first use in this function)  
*Исправление:*  
while (scanf("%d", &num) == 1)
4. *Ошибка:*  
pr\_a.c: В функции «main»:  
pr\_a.c:11:9: ошибка: присваивание, используемое как логическое выражение, рекомендуется [-Werror=parentheses]  
*Исправление:*  
if (num == max)
5. *Ошибка:*  
pr\_a.c: В функции «main»:  
pr\_a.c:14:5: ошибка: format «%d» expects argument of type «int», but argument 2 has type «int \*» [-Werror=format=]  
*Исправление:*  
printf("max %d, count %d\n", max, count);

### Семантические ошибки:

6. Не меняется значение count при нахождении нового максимума  
...  
if (num > max)  
{  
    max = num;  
    count = 1;  
}

7. Если первое число – искомый максимум, то в count будет на 1 меньше, чем есть

```
...
if (scanf("%d", &max) == 1)
    count++;
```

8. Случай, когда чисел нет, не рассмотрен

```
...
int max = 0, count = 0, num, k;
...
if (scanf("%d", &max) == 1)
{
    k++;
    count++;
}
...
if (k>=1)
    printf("max %d, count %d\n", max, count);
else
    printf("There are no numbers!");
...
```

Код, получившийся в результате последовательного исправления семантических и синтаксических ошибок:

```
#include <stdio.h>
int main(void)
{
    int max = 0, count = 0, num, k;
    k = 0;
    if (scanf("%d", &max) == 1)
    {
        k++;
        count++;
    }
    while (scanf("%d", &num) == 1)
    {
        if (num > max)
        {
            max = num;
            count = 1;
        }
        else
            if (num == max)
                count++;
    }

    if (k>=1)
        printf("max %d, count %d\n", max, count);
    else
        printf("There are no numbers!");
    return 0;
}
```

Перед запуском программы в пошаговом режиме для корректной работы отладчика необходимо добавить следующие ключи : **-g3 -gdwarf.**

**gcc -Wall -Werror -pedantic -g3 -gdwarf-2 -c pr\_a.c**

Далее ставим точку останова в нужном нам месте и после остановки при помощи клавиши F10 можно выполнять программу построчно.

Проверим правильность срабатывания точек останова:

Добавим точку останова на условии (num == max) и добавим дополнительное условие (max < 5). При отладке программа будет останавливаться на заданных условиях. На исскомом, когда найден ещё один максимум, а на добавленном, если максимум будет меньше 5.

```
13 while (scanf("%d", &num) == 1)
14 {
15     if (num > max)
16     {
17         max = num;
18         count = 1;
19     }
20     else
21     if (num == max)
22         count++;
23     if (max < 5)
24     {
25         printf("Max < 5");
26     }
27 }
```

При помощи отладчика можно изменить значение переменной во время работы программы в окне переменные и выражения.

Имя	Значение	Тип
co...	1	int
k	1	int
max	5	int
num	3	int

## 5. Индивидуальное задание

### Задание 1.

Пользователь вводит целые числа, по окончании ввода чисел нажимает Ctrl-Z и Enter. Написать программу, которая определяет сколько раз в последовательности чисел меняется знак (нуль считается положительным числом).

*Допущения:* строка всегда завершается буквой

*Псевдокод:*

```
функция main (вход: -; выход: 0)
    int count = 0;
    get_count(stdin, &count)
        обработка ошибочных ситуаций
        вывод(count)
    возврат 0;
функция get_count (вход: fin – файловый указатель, *count – адрес переменной;
выход: 0)
    int tnum, num;
    если (успешное считывание num)
        пока (успешное считывание tnum)
            если ((num положительное) И (tnum отрицательное))
                увеличение на 1 счетчика count;
            иначе
                если ((num < отрицательное) И (tnum положительное))
                    увеличение на 1 счетчика count;
                num = tnum;
        всё пока
    всё если
    иначе
        если (не успешное считывание num: пусто)
            возврат кода ошибки;
        иначе
            если (не успешное считывание num: ошибочные данные)
                возврат кода ошибки;
    возврат 0;
```

*Тесты по классам эквивалентности:*

1. Некорректный ввод:
  - {пустая строка/файл}
  - d d b f
2. Все элементы одного знака:
  - 1 9 6 3 5 a
  - -5 -9 -1 -3 -4 й
3. В строке присутствует смена знака:
  - 2 3 -5 6 -7 0 4 0 -2 q
  - 0 -1 0 5 6 0 -6 r

## Задание 2.

Написать программу, которая считывает из текстового файла вещественные числа и выполняет над ними некоторые вычисления: рассчитать дисперсию чисел (математическое ожидание и дисперсия рассчитываются отдельно).

Математическое ожидание –  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ , дисперсия –  $D = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$ .

*Допущения:* файл содержит только числа, ввод происходит до ввода буквы/символа.

*Псевдокод:*

**функция main**(вход: argc – кол-во аргументов при запуске, argv – массив строк;  
выход: 0)

```
FILE* fin;  
float exp_v;  
float disp;  
exp_v = 0;  
disp = 0;
```

```
проверка кол-ва аргументов при запуске;  
проверка на существование файла, открытие файла(fin);
```

```
expected_value(fin, &exp_v);  
    обработка ошибок функции expected_value;
```

```
обработка файла с начала;  
    disper(fin, &exp_v, &disp);  
обработка ошибок функции disper;  
закрытие файла(fin);  
вывод ответа(disp);
```

```
возврат 0;
```

**функция expected\_value**(вход: fin – входной файл, адрес переменной \*exp\_v; выход: 0)

```
float tmp = 0;  
float x;  
int size = 0;  
int r;
```

```
пока (1)
```

```
    проверка правильности ввода;  
        если ввод некорректен  
            выход из цикла;  
        всё если  
            увеличение счетчика size;  
            добавление текущего x в tmp;
```

```
всё пока
```

```
если данные некорректны  
    возврат кода ошибки;
```

```

всё если
если файл пустой
    возврат кода ошибки;
всё если
(*exp_v) = tmp/size;
возврат 0;

```

**Функция disper**(вход: fin – входной файл, адреса переменных \*exp\_v и \*disp; выход: 0)

```

float x;
float sum_sq = 0;
int r;
int size = 0;

пока (1)
    проверка правильности ввода;
    если ввод некорректен
        выход из цикла;
    всё если
        увеличение счетчика size;
        добавление текущего x*x в sum_sq;
всё пока
(*disp) = (sum_sq-size*(*exp_v)*(*exp_v))/size;
если данные некорректны
    возврат кода ошибки;
всё если
если файл пустой
    возврат кода ошибки;
всё если
возврат 0;

```

*Тесты по классам эквивалентности:*

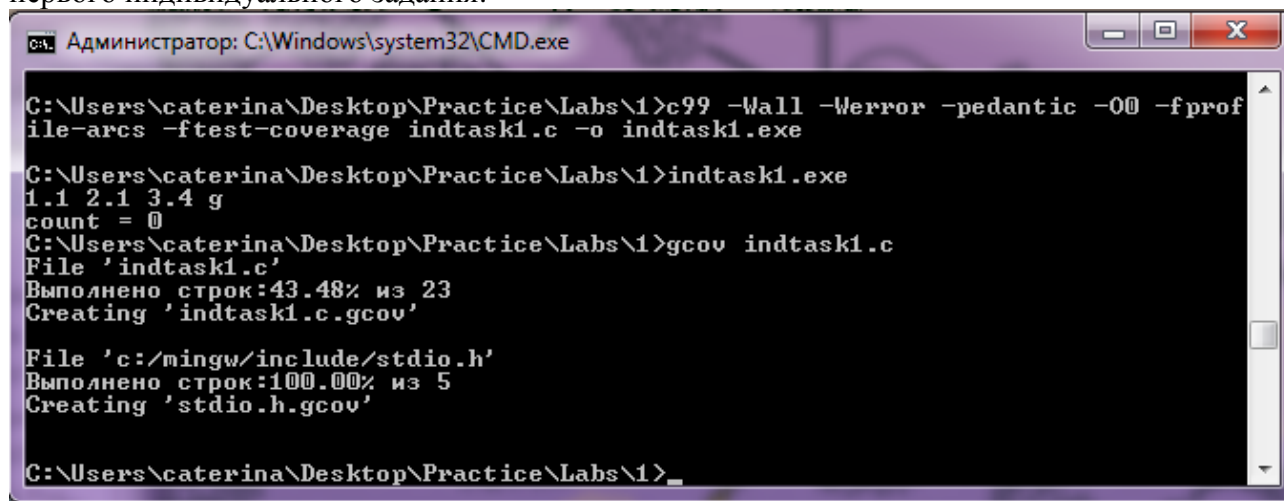
1. Некорректные данные в файле:
  - D s f
  - {пустой файл}
2. Корректные данные в файле:
  - 0.5 2.5 3
  - 1.1 2.1 3.4



## 6. Исследование покрытия кода тестами

Утилита gcov помогает произвести оценку покрытия кода тестами.

При помощи командной строки и следующих команд мы добились 100% покрытия для первого индивидуального задания:



```
Администратор: C:\Windows\system32\CMD.exe

C:\Users\caterina\Desktop\Practice\Labs\1>c99 -Wall -Werror -pedantic -O0 -fprofile-arcs -ftest-coverage indtask1.c -o indtask1.exe

C:\Users\caterina\Desktop\Practice\Labs\1>indtask1.exe
1.1 2.1 3.4 g
count = 0
C:\Users\caterina\Desktop\Practice\Labs\1>gcov indtask1.c
File 'indtask1.c'
Выполнено строк:43.48% из 23
Creating 'indtask1.c.gcov'

File 'c:/mingw/include/stdio.h'
Выполнено строк:100.00% из 5
Creating 'stdio.h.gcov'

C:\Users\caterina\Desktop\Practice\Labs\1>_
```

Количество запусков программы:

-: 0:Runs:1

Произведем оценку покрытия для второго задания:

*Для начала скомпилируем код:*

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>C:\MinGW\bin\gcc -Wall -Werror -O0 -fprofile-arcs -ftest-coverage -o main.exe main.c func.c
```

*Вызовем утилиту gcov для каждого из исходных файлов:*

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>main.exe fin1.txt
```

```
Dispersion = 0.8867
C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov main.c
File 'main.c'
Выполнено строк:40.74% из 27
Creating 'main.c.gcov'
```

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov func.c
File 'func.c'
Выполнено строк:86.67% из 30
Creating 'func.c.gcov'
```

*Откроем файл с расширением .gcov и посмотрим на «аннотированный» листинг.*

*Исправим: #####: 30: return EMPTY;*

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>main.exe fin2.txt
Empty
C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov main.c
File 'main.c'
Выполнено строк:51.85% из 27
Creating 'main.c.gcov'

C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov func.c
File 'func.c'
Выполнено строк:90.00% из 30
Creating 'func.c.gcov'
```

Исправим: #####: 27: return ERROR\_DATA;

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>main.exe fin3.txt
Error data
C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov main.c
File 'main.c'
Выполнено строк:62.96% из 27
Creating 'main.c.gcov'

C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov func.c
File 'func.c'
Выполнено строк:93.33% из 30
Creating 'func.c.gcov'
```

Исправим: #####: 16: fprintf(stderr, "You should give me a file name!");

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>main.exe
You should give me a file name!
C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov main.c
File 'main.c'
Выполнено строк:70.37% из 27
Creating 'main.c.gcov'

C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov func.c
File 'func.c'
Выполнено строк:93.33% из 30
Creating 'func.c.gcov'
```

Исправим: #####: 23: fprintf(stderr, "File doesn't exist");

```
C:\Users\caterina\Desktop\Practice\Labs\2\c>main.exe abacaba.txt
File doesn't exist
C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov main.c
File 'main.c'
Выполнено строк:77.78% из 27
Creating 'main.c.gcov'

C:\Users\caterina\Desktop\Practice\Labs\2\c>gcov func.c
File 'func.c'
Выполнено строк:93.33% из 30
Creating 'func.c.gcov'
```

В данном случае 100% покрытия добиться невозможно, так как при повторном просмотре не могут возникнуть следующие ошибки, потому что, если они есть, то они уже будут обнаружены при первом просмотре файла:

```
-: 44: case EMPTY:
#####: 45:     fprintf(stderr, "Empty");
#####: 46:     fclose(fin);
#####: 47:     return -1;
-: 48:     break;
-: 49: case ERROR_DATA:
#####: 50:     fprintf(stderr, "Error data");
#####: 51:     fclose(fin);
#####: 52:     return -1;
-: 53:     break;
```

```
1: 60: 1T (size == 0 && r == 0)
#####: 61:     return ERROR_DATA;
-: 62:
1: 63: if (size == 0 && r == -1)
#####: 64:     return EMPTY;
-: 65:
1: 66: return 0;
```

Обойтись без их обработки нельзя, так как функция должна быть универсальной. Вывод: 100% покрытия добиться невозможно.

## Заключение

1. Мною изучены стадии компиляции программы, я научилась компилировать и компоновать программу в командной строке (однофайловый и многофайловый проекты)
2. Мною получено представление об организации объектных и исполняемых файлов, я научилась анализировать информацию, которая в них содержится.
3. Я познакомилась с интегрированной средой разработки Qt Creator. Научилась следующему:
  - создавать проекты в Qt Creator;
  - настраивать сборки проектов (release и debug);
  - отлаживать программы в этой среде.
4. Я познакомилась с утилитой make и научилась ее использовать для автоматизации сборки проектов в командной строке, так и в среде Qt Creator.
5. Я познакомилась с утилитой gcov. Научилась определять величину покрытия кода тестами.
6. Мною изучено и закреплено на практике следующее:
  - работа с текстовыми файлами;
  - обработка ошибок;
  - работа с аргументами командной строки.