

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



Факультет Информатика и системы управления

Кафедра Программное обеспечение ЭВМ и информационные технологии

Отчет по учебной практике

Студент Медведев Алексей Вячеславович
(фамилия, имя, отчество)

Группа: ИУ7-21

Преподаватель: Ломовской И.В.

Подпись: _____

Москва, 2016

1. Установка программного обеспечения. Установка Qt Creator.....	3
2. Этапы компиляции программы.....	5
3. Исследование исполняемого файла.....	8
4. Работа с QT Creator.....	13
5. Индивидуальное задание.....	19
6. Исследование покрытия кода тестами.....	22
Заключение.....	24

1. Установка программного обеспечения. Установка Qt Creator

1) Для выполнения данной практической работы была использована операционная система Ubuntu 16.04. С сайта <https://www.qt.io/ru/download-open-source/> был скачан файл-установщик [Qt Creator 3.5.1 for Linux/X11 64-bit \(96 MB\)](#)

Qt Creator

Среда разработки Qt Creator 3.5.1 включена в стандартный бинарный пакет Qt 5.5.1. Если Вам необходим отдельный инсталлятор, пожалуйста, выберите из списка операционную систему, чтобы установить последнюю версию Qt Creator на свой компьютер.

- › Qt Creator 3.5.1 for Windows (79 MB) (info)
- › Qt Creator 3.5.1 for Linux/X11 32-bit (103 MB) (info)
- › Qt Creator 3.5.1 for Linux/X11 64-bit (96 MB) (info)
- › Qt Creator 3.5.1 for Mac (84 MB) (info)

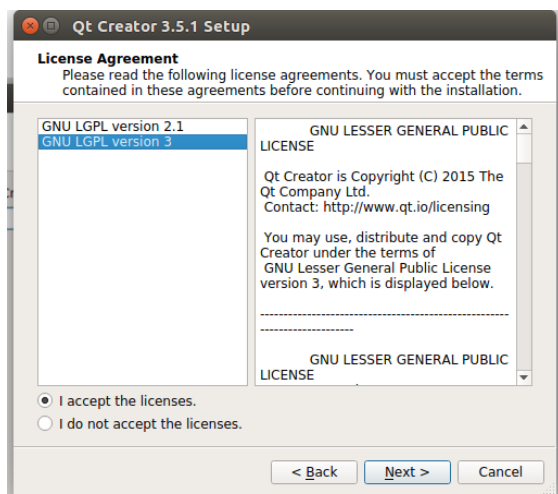
Исходный код доступен в виде [zip-архива](#) (30 MB) (info) или файла [tar.gz](#) (22 MB) (info). Вы также можете посетить репозиторий: code.qt.io..

2) После скачивания файла установочного файла ему были назначены права на исполнение. В терминале выполнить команду: **chmod +x qt-creator-opensource-linux-x86_64-3.5.1.run**

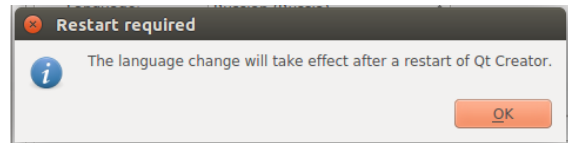
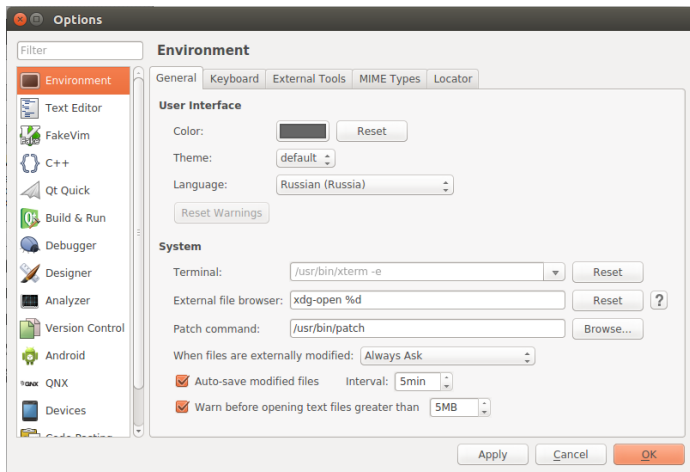
3) Запускаем установочный файл командой в терминале: **./qt-creator-opensource-linux-x86_64-3.5.1.run**

```
alexey@alexey-ub16:~/Documents$ ls -lah
total 97M
drwxr-xr-x  2 alexey alexey 4,0K июл  7 18:10 .
drwxr-xr-x  56 alexey alexey 4,0K июл  7 17:22 ..
-rw-rw-r--  1 alexey alexey  97M июл  5 11:32 qt-creator-opensource-linux-x86_64-3.5.1.run
alexey@alexey-ub16:~/Documents$ chmod +x qt-creator-opensource-linux-x86_64-3.5.1.run
alexey@alexey-ub16:~/Documents$ ls -lah
total 97M
drwxr-xr-x  2 alexey alexey 4,0K июл  7 18:10 .
drwxr-xr-x  56 alexey alexey 4,0K июл  7 17:22 ..
-rwxrwxr-x  1 alexey alexey  97M июл  5 11:32 qt-creator-opensource-linux-x86_64-3.5.1.run
alexey@alexey-ub16:~/Documents$
```

4) Выполняем базовую настройку QT Creator. Проблем с установкой не возникло.



5) После завершения установки запускаем QT Creator, изменяем язык на русский и перезагружаем программу.



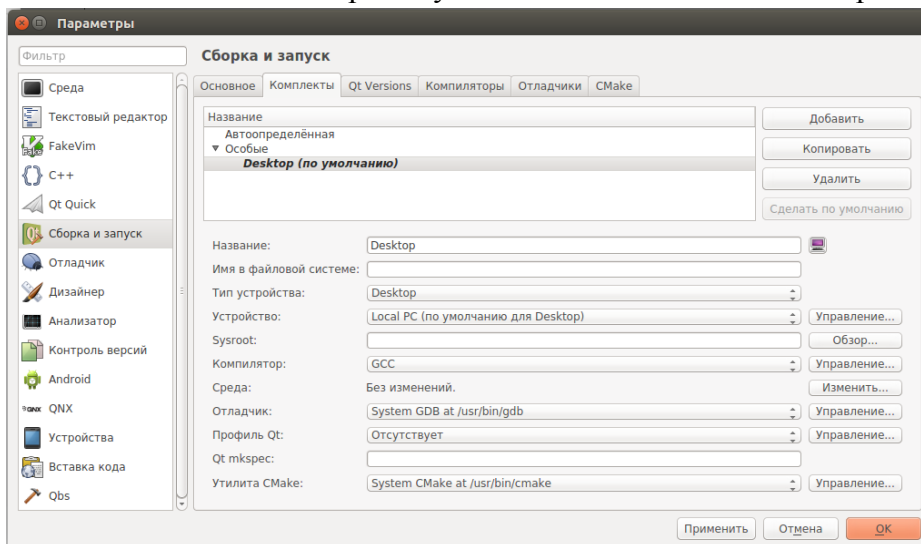
Проверка GDB

Проверяем, что в системе уже установлен GDB и он поддерживает python при помощи двух команд в терминале. Если выводится 1, значит, GDB скомпилирован с поддержкой python.

```
alexey@alexey-ub16:~$ gdb --ex "python import os" --batch 2>&1 | grep Python
alexey@alexey-ub16:~$ echo $?
1
```

Настройка поддержки GCC компилятора и комплектов

Следуя методическим указаниям можно легко настроить GCC компилятор. А в комплектах достаточно просто указать на использование настроенного нами компилятора



2. Этапы компиляции программы

а) Напишем программу

```
//первые 10 чисел фибоначи отличные от 1 1
#include <stdio.h>
#define COUNT 10

int main(void)
{
    int first=1,second=1;

    //вычисляем COUNT чисел
    for (int i = 0; i < COUNT; ++i)
    {
        first=first+second;
        second=first-second;
        print(first);
    }
    return 0;
}
```

Пусть наша программа печатает первые 10 чисел Фибоначи отличных от 1

б) Компилируем ее:

Шаг 1. Обработка препроцессором

Выполним команду **cpp -o main.i main.c**. В результате выполнения данной команды будет создан файл `main.i`, который называется единицей трансляции. На этом этапе препроцессор вырезает комментарии и выполняет текстовые замены директив `define`

```
.....|
extern int dprintf (int __fd, const char *__restrict __fmt, ...)
    __attribute__ ((__format__ (__printf__, 2, 3)));
.....

# 6 "main.c"
int work=1;
int myarray[]={1,2,3,4};
void print(int for_print)
{
    printf("%d\n",for_print );
}

int main(void)
{
    int first=1,second=1;

    for (int i = 0; i < 10; ++i)
    {
        first=first+second;
        second=first-second;
        print(first);
    }
    return 0;
}
```

Шаг 2. Трансляция на язык Ассемблера

Компилятор выполняет трансляцию программы, написанной на Си, на язык ассемблера. Для этого необходимо выполнить команду **c99 -S -masm=intel main.i**. Будет создан файл **main.s** с кодом на языке ассемблер.

```

.....
.LC0:
    .string "%d\n"
    .text
    .globl main
    .type    main, @function

main:
.LFB0:
    .cfi_startproc
    push    rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    mov     rbp, rsp
    .cfi_def_cfa_register 6
    sub     rsp, 16
    mov     DWORD PTR [rbp-12], 1
    mov     DWORD PTR [rbp-8], 1
    mov     DWORD PTR [rbp-4], 0
    jmp     .L2
.....

```

Шаг 3. Ассемблирование в объектный файл

Ассемблер выполняет перевод программы на языке ассемблера в исполнимый машинный код. В результате работы ассемблера получается объектный файл. После выполнения команды **as -o main.o main.s** будет создан файл main.o.

[illegible]

Шаг 4. Компоновка

Компоновщик собирает из объектного файла исполнятельный. Для этого необходимо выполнить команду **ld main.o -o main** и указать необходимые библиотеки. Для того что бы узнать какие библиотеки подключать, необходимо выполнить команду **gcc -v -o main main.o** и скопировать все ключи идущие после -plugin

```
COLLECT_GCC_OPTIONS='-v' '-o' 'main' '-mtune=generic' '-march=x86-64'
```

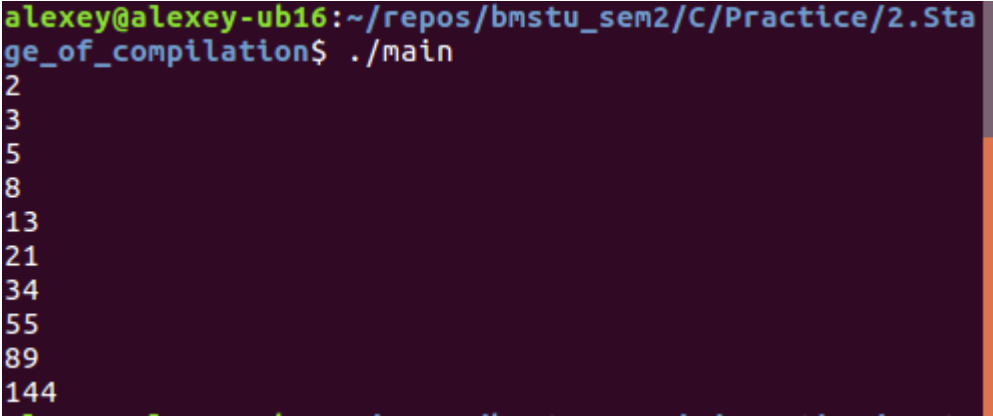
```
COLLECT_GCC_OPTIONS='-v' '-o' 'main' '-mtune=generic' '-march=x86-64'
/usr/lib/gcc/x86_64-linux-gnu/5/collect2 -plugin /usr/lib/gcc/x86_64-linux-gnu/5/li
blto-plugin.so -plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper -plugin-opt=
fresolution=/tmp/ccxkvBjb.res -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-thro
ugh=-lgcc_s -plugin-opt=-pass-through=-lc -plugin-opt=-pass-through=-lgcc -plugin-op
te=-pass-through=-lgcc_s --sysroot=/ --build-id --eh-frame-hdr -m elf_x86_64 --hash-s
tyle=gnu --as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -z relro -o main /u
sr/lib/gcc/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-l
inux-gnu/5/../../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/5/crtbegin
.o -L/usr/lib/gcc/x86_64-linux-gnu/5 -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../../x86
64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../../lib -L/lib64-linux-g
nu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-
linux-gnu/5/../../../../main.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as
-needed -lgcc_s --no-as-needed /usr/lib/gcc/x86_64-linux-gnu/5/crtend.o /usr/lib/gcc
/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crtn.o
```

В итоге наша команда компоновки будет выглядеть следующим образом:

```
ld -plugin /usr/lib/gcc/x86_64-linux-gnu/5/liblto_plugin.so -plugin-  
opt=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper -plugin-opt=-  
fresolution=/tmp/ccxKvBjb.res -plugin-opt=-pass-through=lgcc -plugin-opt=-pass-  
through=lgcc_s -plugin-opt=-pass-through=lc -plugin-opt=-pass-through=lgcc  
-plugin-opt=-pass-through=lgcc_s --sysroot=/ --build-id --eh-frame-hdr -m elf_x86_64  
--hash-style=gnu --as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -z relro -o  
main /usr/lib/gcc/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crt1.o  
/usr/lib/gcc/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-  
linux-gnu/5/crtbegin.o -L/usr/lib/gcc/x86_64-linux-gnu/5 -L/usr/lib/gcc/x86_64-linux-  
gnu/5/../../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../../lib  
-L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib  
-L/usr/lib/gcc/x86_64-linux-gnu/5/../../../../ main.o -lgcc --as-needed -lgcc_s --no-as-needed  
-lc -lgcc --as-needed -lgcc_s --no-as-needed /usr/lib/gcc/x86_64-linux-gnu/5/crtend.o  
/usr/lib/gcc/x86_64-linux-gnu/5/../../../../x86_64-linux-gnu/crtn.o
```

Шаг 5. Запуск

В результате прошлых шагов получаем исполнимый файл main. Запускаем его для теста, набрав в терминале «./main» И в терминале будет следующий вывод:



```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ ./main  
2  
3  
5  
8  
13  
21  
34  
55  
89  
144
```

3. Исследование исполняемого файла

Секции

Полезной функцией утилиты `objdump` является возможность дизассемблировать файл (ключ «-D»). Более того, если объектный файл содержит отладочную информацию – можно дополнительно указать ключ «-S», и утилита выдаст «смешанный код»: строки исходного кода + соответствующие дизассемблированные команды. Ключ «-Н» позволяет узнать какие секции входят в программу.

Скомпилируем две версии программы, с отладочной информацией и без. Сделаем это командами:

```
gcc -std=c99 -pedantic -Wall -Werror main.c -o releasemain
```

```
gcc -std=c99 -pedantic -Wall -Werror main.c -g3 -o debugmain
```

Сравним выводы команды `objdump -D -S` для каждого из исполнительных файлов

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ objdump -D -S debugmain | grep main.: -A30
0000000000400526 <main>:
//первые 10 чисел фибоначи отличные от 1 1
#include <stdio.h>
#define COUNT 10

int main(void)
{
400526:    55                push    %rbp
400527:    48 89 e5          mov     %rsp,%rbp
40052a:    48 83 ec 10       sub     $0x10,%rsp
        int first=1,second=1;
40052e:    c7 45 f4 01 00 00 00 movl    $0x1,-0xc(%rbp)
400535:    c7 45 f8 01 00 00 00 movl    $0x1,-0x8(%rbp)

        //вычисляем COUNT чисел
        for (int i = 0; i < COUNT; ++i)
40053c:    c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp)
400543:    eb 27            jmp     40056c <main+0x46>
        {
400545:    8b 45 f8          mov     -0x8(%rbp),%eax
400548:    01 45 f4          add     %eax,-0xc(%rbp)
        second=first-second;
40054b:    8b 45 f4          mov     -0xc(%rbp),%eax
40054e:    2b 45 f8          sub     -0x8(%rbp),%eax
400551:    89 45 f8          mov     %eax,-0x8(%rbp)
        printf("%d\n",first );
400554:    8b 45 f4          mov     -0xc(%rbp),%eax
400557:    89 c6            mov     %eax,%esi
400559:    bf 04 06 40 00    mov     $0x400604,%edi
40055e:    b8 00 00 00 00    mov     $0x0,%eax

alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ objdump -D -S releasemain | grep main.: -A30
0000000000400526 <main>:
400526:    55                push    %rbp
400527:    48 89 e5          mov     %rsp,%rbp
40052a:    48 83 ec 10       sub     $0x10,%rsp
40052e:    c7 45 f4 01 00 00 00 movl    $0x1,-0xc(%rbp)
400535:    c7 45 f8 01 00 00 00 movl    $0x1,-0x8(%rbp)
40053c:    c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp)
400543:    eb 27            jmp     40056c <main+0x46>
400545:    8b 45 f8          mov     -0x8(%rbp),%eax
400548:    01 45 f4          add     %eax,-0xc(%rbp)
40054b:    8b 45 f4          mov     -0xc(%rbp),%eax
40054e:    2b 45 f8          sub     -0x8(%rbp),%eax
400551:    89 45 f8          mov     %eax,-0x8(%rbp)
400554:    8b 45 f4          mov     -0xc(%rbp),%eax
400557:    89 c6            mov     %eax,%esi
400559:    bf 04 06 40 00    mov     $0x400604,%edi
40055e:    b8 00 00 00 00    mov     $0x0,%eax
400563:    e8 98 fe ff ff    callq   400400 <printf@plt>
400568:    43 45 fc 01       addl    $0x1,-0x4(%rbp)
40056c:    83 7d fc 09       cmpl    $0x9,-0x4(%rbp)
400570:    7e d3            jle     400545 <main+0x1f>
400572:    b8 00 00 00 00    mov     $0x0,%eax
400577:    c9              leaveq  %eax
400578:    c3              retq
400579:    0f 1f 80 00 00 00 00 nopl    0x0(%rax)
```

Версия, скомпилированная с флагом `-g3`, содержит комментарии и исходный код в исполнительном файле, а программа, скомпилированная без этого флага, содержит только код на

языке ассемблер. Причем исполняемые файлы также сильно различаются по размеру — примерно в 3 раза

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ ls -lah debugmain releasemain
-rwxrwxr-x 1 alexey alexey 29K июл  7 20:28 debugmain
-rwxrwxr-x 1 alexey alexey 8,4K июл  7 20:28 releasemain
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$
```

Обе программы содержат общие секции (objdump -h):

interp	rela.plt	init_array
note.ABI-tag	init	fini_array
note.gnu.build-id	plt	jcr
gnu.hash	plt.got	dynamic
dynsym	text	got
dynstr	fini	got.plt
gnu.version	rodata	data
gnu.version_r	eh_frame_hdr	bss
rela.dyn	eh_frame	comment

Версия с отладочной информацией включает в себя еще и секции:

debug_aranges

debug_info debug_abbrev

debug_line

debug_str

debug_macro

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ objdump -D debugmain2 | grep section >debsec
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ objdump -D releasemain | grep section >relsec
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ diff debsec relsec
28,33d27
< Disassembly of section .debug_aranges:
< Disassembly of section .debug_info:
< Disassembly of section .debug_abbrev:
< Disassembly of section .debug_line:
< Disassembly of section .debug_str:
< Disassembly of section .debug_macro:
```

d) В каких секциях находятся переменные?

Для выполнения этого задания добавим в код функцию и глобальную переменную.

```
//первые 10 чисел фибоначи отличные от 1 1
#include <stdio.h>
#define COUNT 10

int work=1;

void print(int for_print)
{
    printf("%d\n",for_print );
}

int main(void)
{
    int first=1,second=1;

    //вычисляем COUNT чисел
    for (int i = 0; i < COUNT; ++i)
    {
        first=first+second;
        second=first-second;
        print(first);
    }
    return 0;
}
```

Выполняем команду `objdump` с флагом `-D` и увидим, что функции находятся в секции `text`. Локальные переменные создаются в оперативной памяти.

```

400420: 11 23 02 00 20 00      jmpq    0x200002(%rip) # 000420
400426: 66 90                  xchg    %ax,%ax

Disassembly of section .text:

0000000000400430 <_start>:
400430: 31 ed                  xor     %ebp,%ebp
400432: 49 89 d1               mov     %rdx,%r9
400435: 5e                    pop     %rsi
400436: 48 89 e2               mov     %rsp,%rdx
400439: 48 83 e4 f0            and     $0xfffffffffffffff0,%rsp
40043d: 50                    push    %rax
40043e: 54                    push    %rsp
400446: 49 c7 c0 10 06 40 00   mov     $0x400610,%r8
40044f: 48 c7 c1 a0 05 40 00   mov     $0x4005a0,%rcx
400454: 48 c7 c7 48 05 40 00   mov     $0x400548,%rdi
400459: e8 b7 ff ff ff        callq   400410 <__libc_start_main@plt>
400459: f4                    hlt

--
0000000000400526 <print>:
400526: 55                    push    %rbp
400527: 48 89 e5               mov     %rsp,%rbp
40052a: 48 83 ec 10            sub     $0x10,%rsp
40052e: 89 7d fc               mov     %edi,-0x4(%rbp)
400531: 8b 45 fc               mov     -0x4(%rbp),%eax
400534: 89 c6                  mov     %eax,%esi
400536: bf 24 06 40 00        mov     $0x400624,%edi
40053b: b8 00 00 00 00        mov     $0x0,%eax
400540: e8 bb fe ff ff        callq   400400 <printf@plt>
400545: 90                    nop
400546: 48                    inc     %eax

```

Инициализированные глобальные переменные — в `.data`

```

Disassembly of section .data:

0000000000601028 <__data_start>:
...

0000000000601030 <__dso_handle>:
...

0000000000601038 <work>:
601038: 01 00                  add     %eax,(%rax)
...

```

е) Глобальный массив. Добавим в код глобальный неинициализированный массив.

```

//первые 10 чисел фибоначи отличные от 1 1
#include <stdio.h>
#define COUNT 10

int work=1;
int myarray[10];
void print(int for_print)
{
    printf("%d\n",for_print );
}

int main(void)
{
    int first=1,second=1;

    //вычисляем COUNT чисел
    for (int i = 0; i < COUNT; ++i)
    {
        first=first+second;
        second=first-second;
        print(first);
    }
    return 0;
}

```

```

Disassembly of section .bss:

0000000000601040 <completed.7585>:
...

0000000000601060 <myarray>:
...

```

Глобальный неинициализированный массив находится в секции `bss`

Размер исполняемого файла увеличился

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ ls -la debugmain*  
-rwxrwxr-x 1 alexey alexey 29368 июл  8 16:17 debugmain2  
-rwxrwxr-x 1 alexey alexey 29456 июл  8 16:17 debugmain3
```

f) Проинициализированный массив

Изменим код, добавив массив.

```
//первые 10 чисел фибоначи отличные от 1 1  
#include <stdio.h>  
#define COUNT 10  
  
int work=1;  
int myarray[]={1,2,3,4};  
void print(int for_print)  
{  
    printf("%d\n",for_print );  
}  
  
int main(void)  
{  
    int first=1,second=1;  
  
    //вычисляем COUNT чисел  
    for (int i = 0; i < COUNT; ++i)  
    {  
        first=first+second;  
        second=first-second;  
        print(first);  
    }  
    return 0;  
}
```

Теперь инициализированный массив находится в раздел data

```
Disassembly of section .data:  
0000000000601030 <__data_start>:  
    ...  
0000000000601038 <__dso_handle>:  
    ...  
0000000000601040 <work>:  
    601040:    01 00                add    %eax,(%rax)  
    ...  
0000000000601050 <myarray>:  
    601050:    01 00                add    %eax,(%rax)  
    601052:    00 00                add    %al,(%rax)  
    601054:    02 00                add    (%rax),%al  
    601056:    00 00                add    %al,(%rax)  
    601058:    03 00                add    (%rax),%eax  
    60105a:    00 00                add    %al,(%rax)  
    60105c:    04 00                add    $0x0,%al  
    60105e:    00 00                add    %al,(%rax)  
    601060:    05 00 00 00 06      add    $0x6000000,%eax  
    601065:    00 00                add    %al,(%rax)  
    ...  
Disassembly of section .bss:  
0000000000601068 <__bss_start>:  
    ...  
Disassembly of section .comment:
```

Размер исполнительного файла еще больше

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ ls -la debugmain*  
-rwxrwxr-x 1 alexey alexey 29368 июл  8 16:17 debugmain2  
-rwxrwxr-x 1 alexey alexey 29456 июл  8 16:17 debugmain3  
-rwxrwxr-x 1 alexey alexey 29496 июл  8 16:18 debugmain4
```

g) Динамические библиотеки

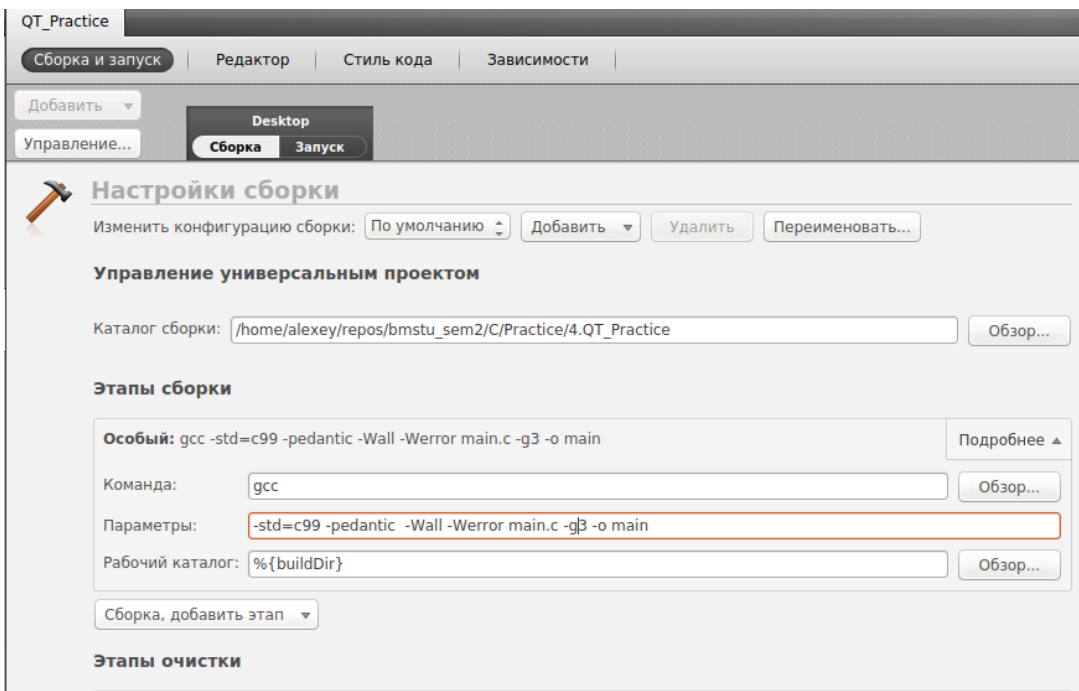
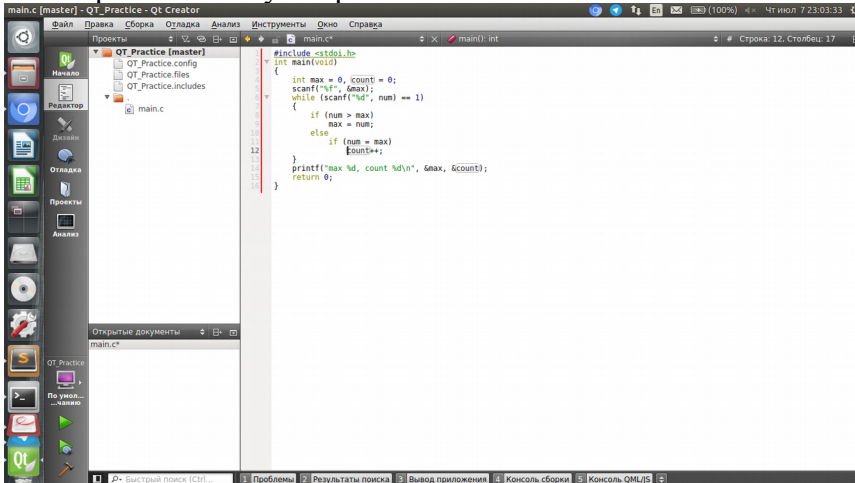
Воспользуемся командой ldd

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/2.Stage_of_compilation$ ldd debugmain4  
linux-vdso.so.1 => (0x00007ffff944b6000)  
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fae1a3dc000)  
/lib64/ld-linux-x86-64.so.2 (0x00005613e8689000)
```

4. Работа с QT Creator

а-в) Создание проект

Создадим файл main.c, с кодом из приложения А.
И настроим систему сборки.



с) Исправние ошибок

I) Синтаксические ошибки

1)



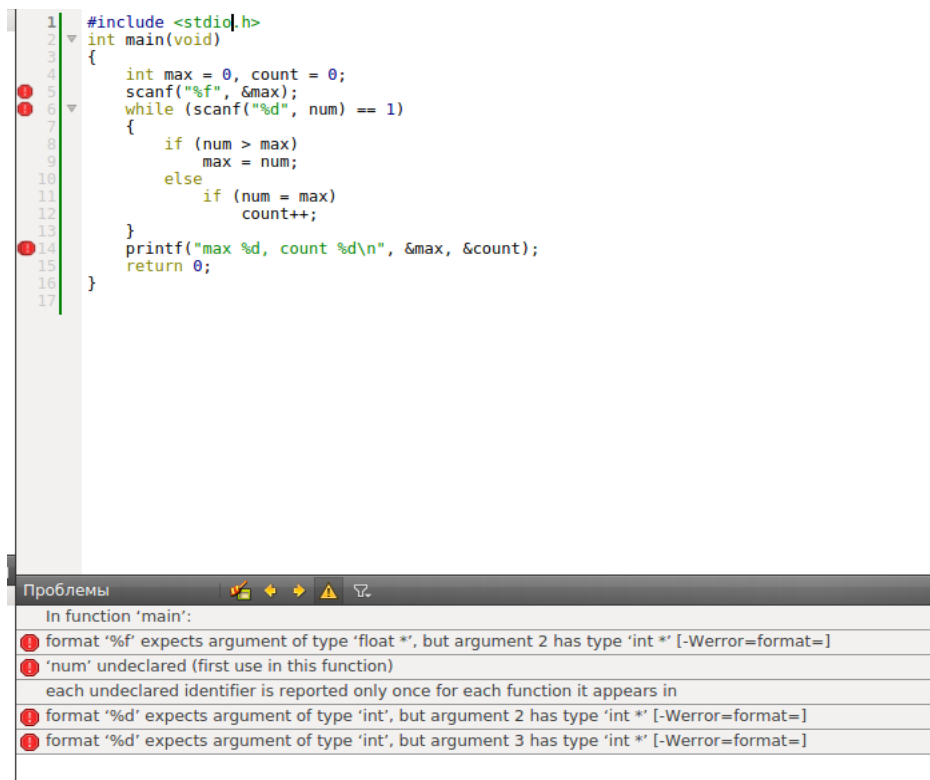
```
1 #include <stdoi.h>
2 int main(void)
3 {
4     int max = 0, count = 0;
5     scanf("%f", &max);
6     while (scanf("%d", num) == 1)
7     {
8         if (num > max)
9             max = num;
10        else
11            if (num = max)
12                count++;
13    }
14    printf("max %d, count %d\n", &max, &count);
15    return 0;
16 }
17
```

Проблемы

stdoi.h: No such file or directory

Исправление #include <stdio.h>

2) ошибка:



```
1 #include <stdio.h>
2 int main(void)
3 {
4     int max = 0, count = 0;
5     scanf("%f", &max);
6     while (scanf("%d", num) == 1)
7     {
8         if (num > max)
9             max = num;
10        else
11            if (num = max)
12                count++;
13    }
14    printf("max %d, count %d\n", &max, &count);
15    return 0;
16 }
17
```

Проблемы

In function 'main':

- format '%f' expects argument of type 'float *', but argument 2 has type 'int *' [-Werror=format=]
- 'num' undeclared (first use in this function)
- each undeclared identifier is reported only once for each function it appears in
- format '%d' expects argument of type 'int', but argument 2 has type 'int *' [-Werror=format=]
- format '%d' expects argument of type 'int', but argument 3 has type 'int *' [-Werror=format=]

format '%f' expects argument of type 'float *', but argument 2 has type 'int *'

Исправление - scanf("%f", &max); заменить scanf("%d", &max);

3) 'num' undeclared (first use in this function)

```
while (scanf("%d", num) == 1)
```

Необходимо объявить переменную num: int num;

В scanf нужно передавать адрес переменной

```
while (scanf("%d", &num) == 1)
```

4) error: suggest parentheses around assignment used as truth value [-Werror=parentheses]

```
if (num = max)
```

Исправление - if (num == max)

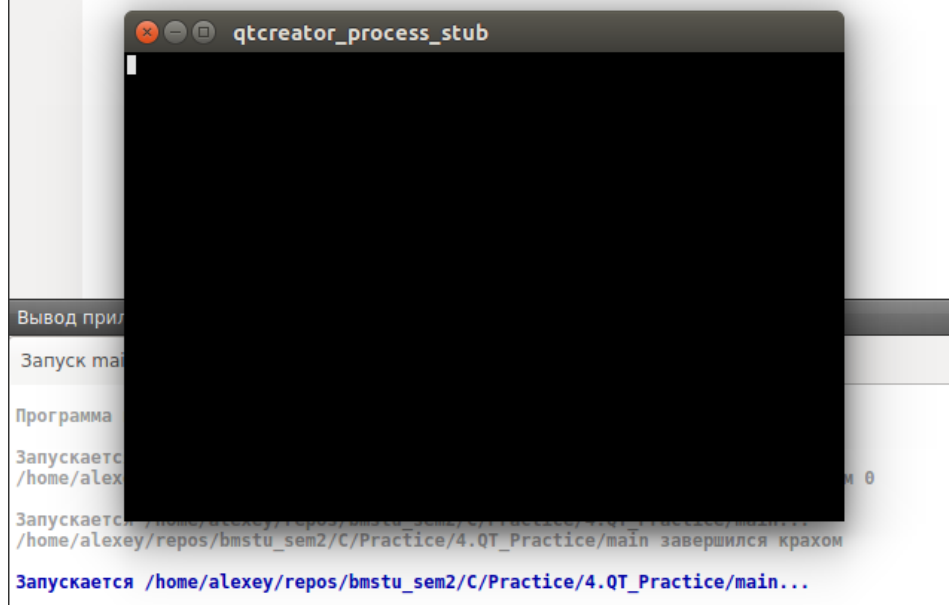
5) ошибка: format '%d' expects argument of type 'int', but argument 3 has type 'int *' [-Werror=format=]

Исправление: в printf нужно передавать саму переменную, а не её адрес

```
printf("max %d, count %d\n", max, count);
```

Теперь программа компилируется.

```
1 #include <stdio.h>
2 int main(void)
3 {
4     int max = 0, count = 0, num;
5     scanf("%d", &max);
6     while (scanf("%d", &num) == 1)
7     {
8         if (num > max)
9             max = num;
10        else
11            if (num == max)
12                count++;
13    }
14    printf("max %d, count %d\n", max, count);
15    return 0;
16 }
17
```



II) Семантические ошибки

1) int max = 0, count = 0;

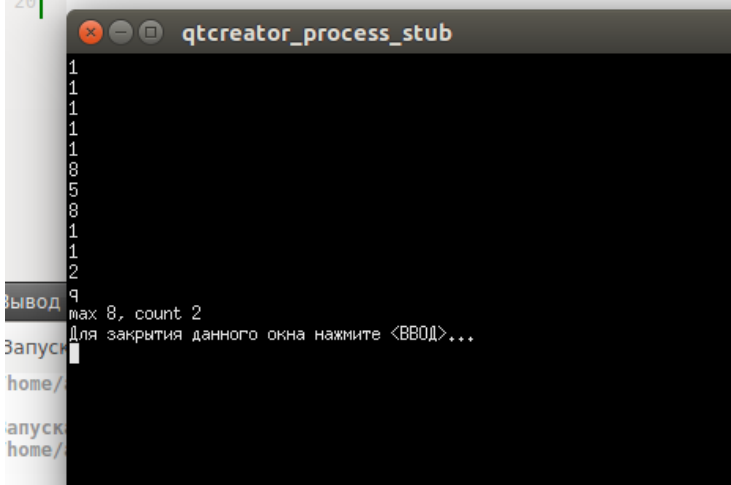
Мы предполагаем, что первое вводимое число является максимальным. Поэтому нужно изменить эту строку на

```
int max = 0, count = 1;
```

2) if (num > max)
 max = num;

Нашли число больше, чем все предыдущие, но не указали, что количество самых больших чисел стало 1. Для этого надо добавить count = 1;

```
1  #include <stdio.h>
2  int main(void)
3  {
4      int max = 0, count = 1, num;
5      scanf("%d", &max);
6      while (scanf("%d", &num) == 1)
7      {
8          if (num > max)
9          {
10             max = num;
11             count=1;
12         }
13         else
14             if (num == max)
15                 count++;
16     }
17     printf("max %d, count %d\n", max, count);
18     return 0;
19 }
26
```

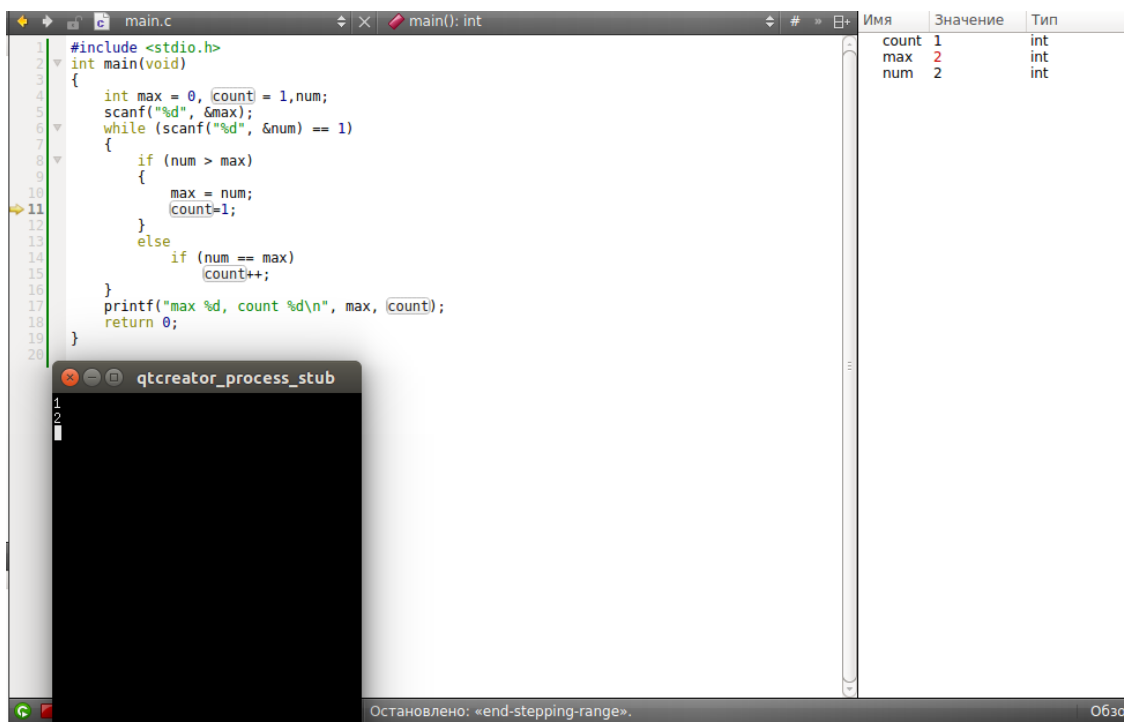


Вывод max 8, count 2
Для закрытия данного окна нажмите <ВВОД>...

Запуск
home/
пуск
home/

Теперь программа работает корректно.

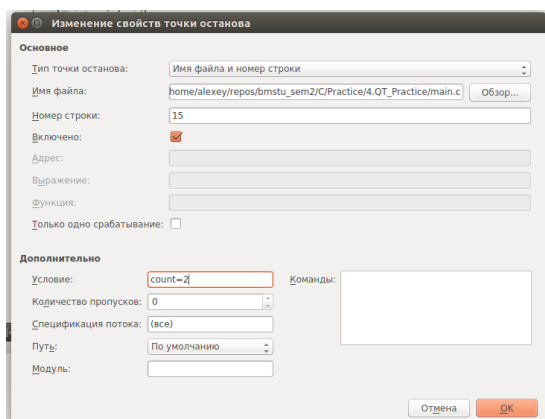
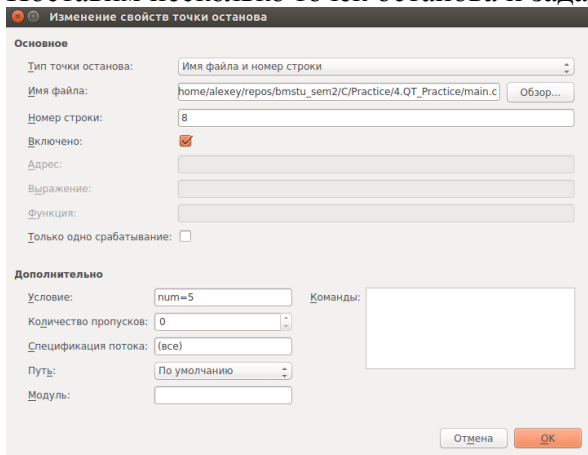
д) Выполним программу в пошаговом режиме

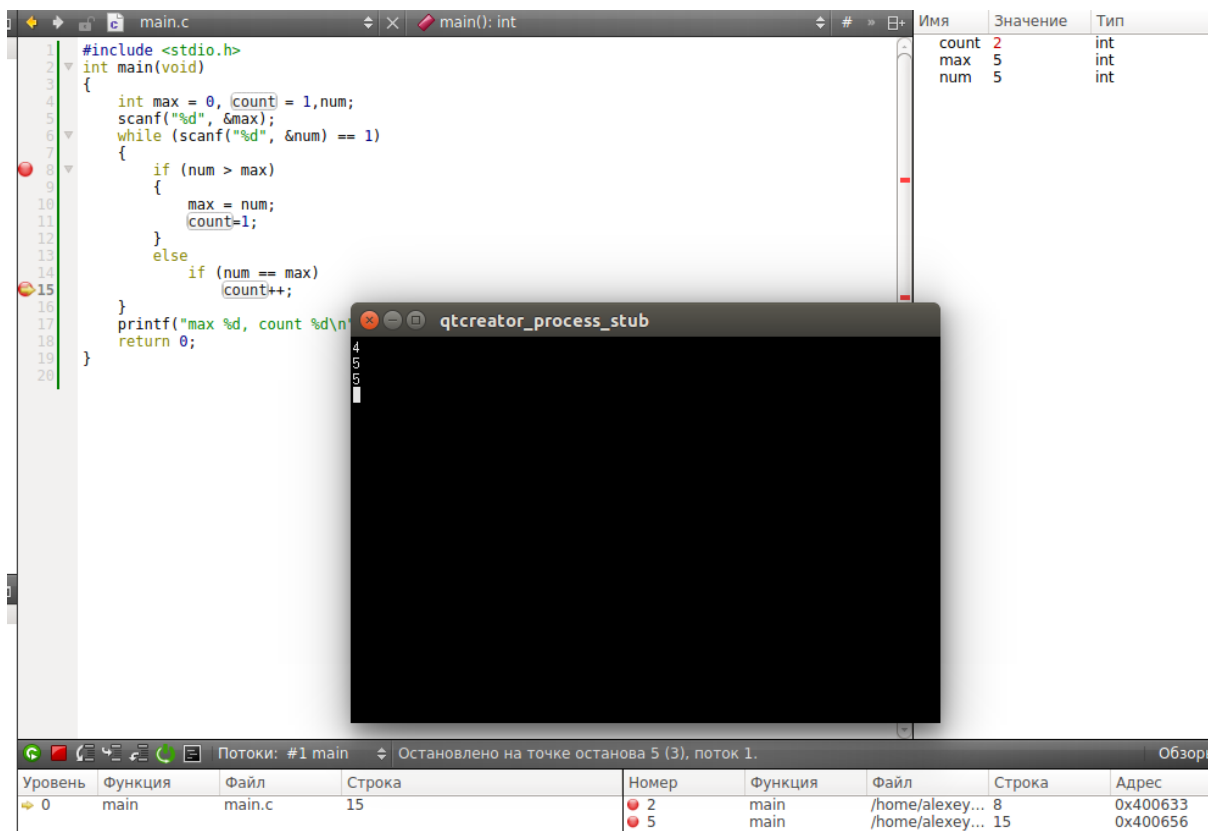


При этом мы видим значения всех переменных

е) Точки останова

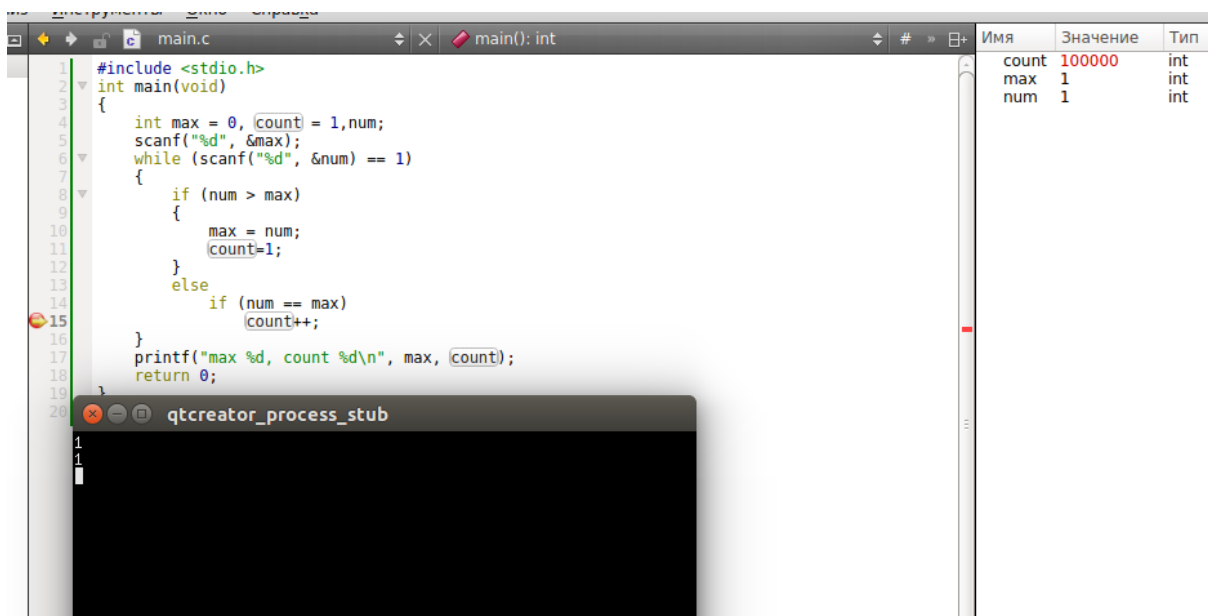
Поставим несколько точек останова и зададим им условия





Условия выполняются и срабатывает точка останова.

f) Изменение переменные во время выполнения программы



Мы ввели данные. Мы видим значения переменных. Попробуем изменить count на 100000

Как видим, значение переменной изменилось.

```

1
1
q
max 1, count 100001
Для закрытия данного окна нажмите <ВВОД>...

```

5. Индивидуальное задание

Задание 1 (вариант 3)

1) Условие

Пользователь вводит целые числа, по окончании ввода чисел нажимает Ctrl-Z и Enter.

Написать программу, которая находит наибольшее число подряд идущих элементов последовательности, которые равны друг другу;

2) Допущения:

- 1) Пользовательский ввод происходит с консоли
- 2) Файл содержит только числа (если встречается не число, то считывание прерывается)
- 3) Вывод происходит на консоль.

3) Классы эквивалентности

- 1) Пустой ввод (ctrl+Z)
- 2) При вводе встречаются символы (не числа)
- 3) Ввод содержит только одно число.
- 4) Все числа одинаковые
- 5) Все числа различные
- 6) Различные цепочки разной длины

4) Тесты:

- 1) Ctrl+z
- 2) 5 a
- 3) 4
- 4) 2 2 2 2 2 2
- 5) 1 2 3 4 5 6
- 6) 1 2 2 2 1 1

5) Псевдокод

```
int current_element = 0;
int element_before = 0;
int max_length_of_identical = 1;
int temp_max_length_of_identical = 1;
```

```
Если пользователь ввел число (current_element)
    element_before = current_element;
Иначе
    return -1
```

```
ПОКА пользователь вводит число (current_element)
    Если current_element == element_before
        temp_max_length_of_identical++
    Иначе
        temp_max_length_of_identical = 1
```

```
Если temp_max_length_of_identical > max_length_of_identical
    max_length_of_identical = temp_max_length_of_identical
    element_before = current_element;
return 0
```

Задание 2 (вариант 2)

1) Условие

Написать программу, которая считывает из текстового файла вещественные числа и рассчитывает дисперсию чисел (математическое ожидание и дисперсия рассчитываются отдельно)

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

2) Допущения:

- 1) Ввод происходит из файла
- 2) Пользователь вводит аргумент командной строки — существующего файл
- 3) Файл содержит только числа (если встречается не число, то считывание прерывается)

3) Классы эквивалентности

- 1) Пустой файл
- 2) Во вводе встречаются символы (не числа)
- 3) Ввод содержит только одно число.
- 4) Ввод содержит несколько чисел

4) Тесты:

- 1) Пустой файл
- 2) 5 а 5
- 3) 5
- 4) 1 2 3 4 -5

5) Псевдокод

Функция expected_value

Вход: файл ввода

Выход: дробное число(expectvalue)

```
float summ=0;
int n=0;
float temp;
Пока можно считать число(temp))
    summ=summ+temp;
    n++;
Если(n==0)
    return ERROR_NE;
expectvalue=summ/n;
return 0;
```

Функция dispersionfunc

Вход: файл ввода, дробное число(expectvalue)

Выход: дробное число(dispersion)

float summ=0;

int n=0;

float temp;

Пока можно считать число(temp))

 summ=summ+(temp-expectvalue)*(temp-expectvalue);

 n++;

Если(n==0)

 return ERROR_NE;

dispersion=summ/n;

return 0;

Проекты называются 5.Multi_file_task 5.Multi_file_task_tests 5.Multi_file_task_headers

6. Исследование покрытия кода тестами

Оцените покрытие кода тестами. Если оно недостаточно расширьте тестовые наборы.

Для первой задачи

Создадим makefile

```
main: main.o
    gcc -o $@ $^

%.o: %.c
    gcc $(FLAGS) -c $<

test:
    ./main <test1
    ./main <test2
    ./main <test3
    ./main <test4
    ./main <test5
    ./main <test6

gcov:
    rm *.gc*
    c99 -Wall -Werror -pedantic -O0 -fprofile-arcs -ftest-coverage *.c -o main
    make -B test
    gcov *.c

clean:
    rm *.o main
    rm *.gc*
```

И запустим цель gcov.

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/5.Single_file_task$ make -B -i gcov
rm *.gc*
rm: cannot remove '*.gc*': No such file or directory
makefile:25: recipe for target 'gcov' failed
make: [gcov] Error 1 (ignored)
c99 -Wall -Werror -pedantic -O0 -fprofile-arcs -ftest-coverage *.c -o main
make -B test
make[1]: Entering directory '/home/alexey/repos/bmstu_sem2/C/Practice/5.Single_file_task'
./main <test1
BAD INPUT./main <test2

Наибольшее число подряд идущих: 1
./main <test3

Наибольшее число подряд идущих: 1
./main <test4

Наибольшее число подряд идущих: 7
./main <test5

Наибольшее число подряд идущих: 1
./main <test6

Наибольшее число подряд идущих: 3
make[1]: Leaving directory '/home/alexey/repos/bmstu_sem2/C/Practice/5.Single_file_task'
gcov *.c
File 'main.c'
Lines executed:100.00% of 27
Creating 'main.c.gcov'
```

В данной задаче удалось добиться полного покрытия тестами.

Для второй задачи

```
alexey@alexey-ub16:~/repos/bmstu_sem2/C/Practice/5.Multi_file_task$ make -i -B gcov
гм *.gc*
c99 -Wall -Werror -pedantic -O0 -fprofile-arcs -ftest-coverage main.c functions.c -o main
make -B -i test
make[1]: Entering directory '/home/alexey/repos/bmstu_sem2/C/Practice/5.Multi_file_task'
./main
Недостаточно аргументовmakefile:18: recipe for target 'test' failed
make[1]: [test] Error 253 (ignored)
./main test2

Математическое ожидание: 5.600000

Дисперсия: 0.000000
./main test3

Математическое ожидание: 5.500000

Дисперсия: 0.000000
./main test4

Математическое ожидание: 3.500000

Дисперсия: 2.420000
./main test5

Математическое ожидание: 5.700000

Дисперсия: 5.835000
./main test1
Недостаточно данныхmakefile:18: recipe for target 'test' failed
make[1]: [test] Error 255 (ignored)
./main not_exist_file
Не могу открыть файл 'not_exist_file'makefile:18: recipe for target 'test' failed
make[1]: [test] Error 254 (ignored)
make[1]: Leaving directory '/home/alexey/repos/bmstu_sem2/C/Practice/5.Multi_file_task'
gcov main.c functions.c
File 'main.c'
Lines executed:100.00% of 28
Creating 'main.c.gcov'

File 'functions.c'
Lines executed:100.00% of 22
Creating 'functions.c.gcov'

Lines executed:100.00% of 50
```

Во второй задаче также удалось добиться полного покрытия кодами тестами.

Заключение

- В рамках практики были получены и закреплены на практике знания о стадиях компиляции и компоновке программы в командной строке.
- Была изучена организация объектных и исполняемых файлов.
- Получены навыки по работе со средой QT Creator, в том числе: настраивать сборки проектов, отлаживать программы.
- Были изучены новые утилиты `make`, `gcov`, `objdump`.
- Также была изучена работа с текстовыми файлами, обработка ошибок, работа с аргументами командной строки.