Отчет по заданию №3 в рамках вычислительного практикума «Отладка»

Студент: Краснов Леонид, группа ИУ7-21Б

Содержание

Цель работы

Целью работы является умение студента самостоятельно производить трассировку приложения.

Задание №1

Найти с помощью gdb ошибки в приложенных программах. Описать как это сделано.

Отладка «task_01.c»

1. **Сборка исполняемого файла**

```
gcc -Wall -Werror -Wpedantic -g -o app.exe task_01.c
```

2. **Запуск исполняемого файла с gdb**

```
kali% gdb ./app.exe
GNU gdb (Debian 10.1-2+b1) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "aarch64-linux-gnu".
--Type <RET> for more, q to quit, c to continue without paging--c
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./app.exe...
```

3. Проверка введенного значения

```
(gdb) b 17
Breakpoint 1 at 0x818: file task_01.c, line 17.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Input n: 10

Breakpoint 1, main () at task_01.c:17
17          result = factorial(n);
(gdb) print n
$1 = 10
```

Таким образом получаем что программа правильно считывает переменную

4. Проверим что передается в функцию factorial()

```
(gdb) b factorial
Breakpoint 1 at 0x850: file task_01.c, line 26.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Input n: 10

Breakpoint 1, factorial (n=10) at task_01.c:26
26          long long unsigned result = 1;
(gdb) print n
$1 = 10
```

Получаем что значение переданное в factorial верно

5. Проверим значение, которое возвращает функция factorial

```
(gdb) b 31
Breakpoint 1 at 0x880: file task_01.c, line 31.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Input n: 10

Breakpoint 1, factorial (n=4294967295) at task_01.c:31
31          return result;
(gdb) print result
$1 = 0
```

Получаем что функция возвращает неверное значение, значит ошибка в ней

6. Проверим как работает функция factorial пошагово

```
(gdb) b factorial
Breakpoint 1 at 0x850: file task_01.c, line 26.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Input n: 10

Breakpoint 1, factorial (n=10) at task_01.c:26
26          long long unsigned result = 1;
(gdb) step
28          while (n--)
(gdb) step
29              result *= n;
(gdb) print result
$1 = 1
(gdb) step
28          while (n--)
(gdb) print result
$2 = 9
...
```

```
(gdb) step
29              result *= n;
(gdb) print n
$8 = 0
(gdb) print result
$9 = 362880
(gdb) step
28          while (n--)
(gdb) print result
$10 = 0
```

Получается, что в результате работы цикла переменная result обнуляется, из-за того что цикл продолжает работать когда переменная n = 0 и не работает, когда n = начальному значению, следовательно, у цикла не верное условие.

7. Исправление ошибок

Что бы исправить работу программы нужно заменить условие

```
(n--) на (n-- - 1)
```

и перед выполнением цикла прибавить к n 1

8. Исправленный код

```
long long unsigned factorial(unsigned n)
{
    long long unsigned result = 1;
    n++;
    while (n-- - 1)
        result *= n;

    return result;
}
```

9. Проверка работы

```
kali% gcc -Wall -Werror -Wpedantic -o app.exe task_01.c
kali% ./app.exe
Input n: 10
factorial(10) = 3628800
kali% ./app.exe
Input n: 3
factorial(3) = 6
```

Отладка «task_02.c»

1. Сборка программы для дальнейшей отладки

```
gcc -std=c99 -Wall -Werror -Wpedantic -g task_02.c -o app.exe
```

2. Запуск сеанса отладки

```
gdb ./app.exe
```

3. Пробный запуск программы

```
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 10
Enter the next number: 29
Enter the next number: 12
Enter the next number: 12
Enter the next number: 23
Value [1] is 23
Value [2] is -135943528
```

```
Value [3] is 65535
Value [4] is -1431696944
The average is 0
The max is -1431696944
[Inferior 1 (process 2784) exited normally]
```

Очевидно, что в массиве все кроме первого числа
мусор, среднеарифметическое и максимум посчитались неправильно. Начнем отладку с записи данных в массив

4. Отладка записи данных

Для отладки записи поставим точку остановы на считывание данных и пошагово пройдем цикл заполнения массива

```
(gdb) b 19
Breakpoint 4 at 0xaaaaaaaa0800: file task_02.c, line 19.
(gdb) c
Continuing.

Breakpoint 4, main () at task_02.c:19
19              if (scanf("%d", &arr[1]) != 1)
(gdb) next
Enter the next number: 1
16          for (i = 0; i < N; i++)
(gdb) next
18              printf("Enter the next number: ");
(gdb) next

Breakpoint 4, main () at task_02.c:19
19              if (scanf("%d", &arr[1]) != 1)
(gdb)
Enter the next number: 2
16          for (i = 0; i < N; i++)
(gdb) print arr
$1 = {-5088, 2, -135943528, 65535, -1431696944}
(gdb) n
18              printf("Enter the next number: ");
(gdb)

Breakpoint 4, main () at task_02.c:19
19              if (scanf("%d", &arr[1]) != 1)
(gdb)
Enter the next number: 3
16          for (i = 0; i < N; i++)
(gdb) print arr
$2 = {-5088, 3, -135943528, 65535, -1431696944}
(gdb)
```

Можно заметить, что ввод происходит во второй элемент массива (под индексом 1), чтобы это исправить надо поменять параметры scanf:

```
scanf("%d", &arr[1]) на scanf("%d", &arr[i])
```

скомпилируем программу и запустим под отладчиком заново

```
kali% gcc -std=c99 -Wall -Werror -Wpedantic -g task_02.c -o app.exe
kali% gdb ./app.exe
GNU gdb (Debian 10.1-2+b1) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
```

```
This GDB was configured as "aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
--Type <RET> for more, q to quit, c to continue without paging--c
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./app.exe...
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 1
Enter the next number: 2
Enter the next number: 3
Enter the next number: 4
Enter the next number: 5
Value [1] is 2
Value [2] is 3
Value [3] is 4
Value [4] is 5
The average is 0
The max is 1
[Inferior 1 (process 3357) exited normally]
```

5. Отладка вывода массива

Можно заметить, что вместо 5-ти элементов массива выводится только 4 Для исправления этой ошибки поставим точку остановы в цикл вывода массива

```
(gdb) b 26
Breakpoint 1 at 0xaaaaaaaa0854: file task_02.c, line 26.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 1
Enter the next number: 2
Enter the next number: 3
Enter the next number: 4
Enter the next number: 5

Breakpoint 1, main () at task_02.c:26
26          for (i = 1; i < N; i++)
(gdb) list
21                  printf("Input error");
22                  return 1;
23              }
24          }
25
26          for (i = 1; i < N; i++)
27              printf("Value [%zu] is %d\n", i, arr[i]);
28
29          printf("The average is %g\n", get_average(arr, N));
--Type <RET> for more, q to quit, c to continue without paging--c
30
(gdb) next
27              printf("Value [%zu] is %d\n", i, arr[i]);
(gdb) next
Value [1] is 2
(gdb) print arr
```

```
$1 = {1, 2, 3, 4, 5}
```

Здесь видно что вывод массива начинается со 2-го элемента, чтобы это исправить нужно поменять условие цикла for:

```
for (i = 1; i < N; i++) на for (i = 0; i < N; i++)
```

Скомпилируем и запустим программу:

```
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 1
Enter the next number: 2
Enter the next number: 3
Enter the next number: 4
Enter the next number: 5
Value [0] is 1
Value [1] is 2
Value [2] is 3
Value [3] is 4
Value [4] is 5
The average is 0
The max is 1
[Inferior 1 (process 3707) exited normally]
```

6. Исправление ошибки в нахождении среднеарифметического

Можно заметить что программа выдала 0 вместо среднеарифметического поставим точку останова в функцию get_average для проверки передаваемого значения и работы функции

```
(gdb) break get_average
Breakpoint 1 at 0xaaaaaaaaa08e4: file task_02.c, line 38.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 1
Enter the next number: 2
Enter the next number: 3
Enter the next number: 4
Enter the next number: 5
Value [0] is 1
Value [1] is 2
Value [2] is 3
Value [3] is 4
Value [4] is 5

Breakpoint 1, get_average (a=0xffffffffec00, n=5) at task_02.c:38
38          double temp = 0.0;
(gdb) print a
$1 = (const int *) 0xffffffffec00
(gdb) print *a
$2 = 1
```

Видно, что параметры передались корректно. Теперь посмотрим что возвращает функция:

```
(gdb) b 44
Breakpoint 2 at 0xaaaaaaaaa0944: file task_02.c, line 44.
(gdb) c
Continuing.

Breakpoint 2, get_average (a=0xffffffffec00, n=5) at task_02.c:44
44          return temp;
```

```
(gdb) print temp
$3 = 0
```

Так как функция принимает верные параметры, а возвращает 0 можно сделать вывод, что ошибка в функции, для ее нахождения пройдем get_average пошагово

```
Breakpoint 1, get_average (a=0xffffffffec00, n=5) at task_02.c:38
38          double temp = 0.0;
(gdb) s
40          for (size_t i = 0; i > n; i++)
(gdb) s
42          temp /= n;
(gdb)
```

Цикл for не срабатывает, значит у него неверное условие. Для корректной работы нужно поменять:

```
for (size_t i = 0; i > n; i++) на for (size_t i = 0; i < n; i++)
```

Скомпилируем и запустим программу:

```
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 1
Enter the next number: 2
Enter the next number: 3
Enter the next number: 4
Enter the next number: 5
Value [0] is 1
Value [1] is 2
Value [2] is 3
Value [3] is 4
Value [4] is 5
The average is 3
The max is 1
[Inferior 1 (process 4048) exited normally]
```

7. Исправление ошибки в нахождении максимума

Видно, что программа находит максимум неверно, для устранения этой ошибки поставим точку останова в функцию get_max для проверки входных и выходных данных и ее работы:

```
(gdb) b get_max
Breakpoint 1 at 0xaaaaaaaa095c: file task_02.c, line 49.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 1
Enter the next number: 2
Enter the next number: 5
Enter the next number: 4
Enter the next number: 3
Value [0] is 1
Value [1] is 2
Value [2] is 5
Value [3] is 4
Value [4] is 3
The average is 3

Breakpoint 1, get_max (a=0xffffffffec00, n=5) at task_02.c:49
49          int max = a[0];
```

```
(gdb) l
44          return temp;
45      }
46
47      int get_max(const int *a, size_t n)
48      {
49          int max = a[0];
50
51          for (size_t i = 1; i < n; i++)
52              if (max > a[i])
--Type <RET> for more, q to quit, c to continue without paging--c
53                  max = a[i];
(gdb) l
54
55          return max;
56      }
(gdb) b 55
Breakpoint 2 at 0xaaaaaaaa09c8: file task_02.c, line 55.
(gdb) c
Continuing.

Breakpoint 2, get_max (a=0xffffffffec00, n=5) at task_02.c:55
55          return max;
(gdb) print max
$1 = 1
```

Входные параметры корректны, а на выходе неверный результат - значит функция работает некорректно для нахождения и устранения ошибки пройдем функцию пошагово

```
Breakpoint 1, get_max (a=0xffffffffec00, n=5) at task_02.c:49
49          int max = a[0];
(gdb) n
51          for (size_t i = 1; i < n; i++)
(gdb) n
52              if (max > a[i])
(gdb) p a[i]
$2 = 2
(gdb) p max
$3 = 1
(gdb) n
51          for (size_t i = 1; i < n; i++)
```

Здесь видно что условие не выполняется из-за его некорректности и вместо максимума находится минимум, что бы это исправить нужно заменить:

```
if (max > a[i]) на if (max < a[i])
```

Скомпилируем и запустим:

```
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
Enter 5 numbers:
Enter the next number: 3
Enter the next number: 2
Enter the next number: 5
Enter the next number: 1
Enter the next number: 4
Value [0] is 3
Value [1] is 2
Value [2] is 5
Value [3] is 1
Value [4] is 4
```

```
The average is 3
The max is 5
[Inferior 1 (process 4293) exited normally]
```

Все ошибки исправлены, и программа работает корректно

Отладка «task_03.c»

1. **Сборка программы для дальнейшей отладки**

```
gcc -std=c99 -Wall -Werror -Wpedantic -g task_03.c -o app.exe
```

2. **Запуск сеанса отладки**

```
gdb ./app.exe
```

3. **Пробный запуск программы**

```
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
5 div 2 = 2
10 div 0 = 0
[Inferior 1 (process 2162) exited normally]
```

Программа работает корректно. Выполним ее пошагово

```
(gdb) b main
Breakpoint 1 at 0x7dc: file task_03.c, line 7.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe

Breakpoint 1, main () at task_03.c:7
7           int a = 5, b = 2;
(gdb) s
9           printf("%d div %d = %d\n", a, b, div(a, b));
(gdb)
div (numer=5, denom=2) at div.c:58
58      div.c: Нет такого файла или каталога.
(gdb)
61      in div.c
(gdb)
```

Здесь видно, что программа использует функцию div из сторонней библиотеки, а не из программы, это из-за опечатки в объявлении функции div, вместо этого функция называется dib. Исправим, соберем и запустим программу:

```
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe
5 div 2 = 2
10 div 0 = 0
[Inferior 1 (process 2479) exited normally]
```

Программа сработала корректно, чтобы убедится что программа использует функцию div, описанную в программе поставим точку остановы внутрь этой функции и запустим.

```
(gdb) b div
Breakpoint 1 at 0x818: file task_03.c, line 21.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice3/app.exe

Breakpoint 1, div (a=5, b=2) at task_03.c:21
21          return a / b;
(gdb) c
Continuing.
5 div 2 = 2
```

```
Breakpoint 1, div (a=10, b=0) at task_03.c:21
21          return a / b;
(gdb)
Continuing.
10 div 0 = 0
[Inferior 1 (process 1930) exited normally]
(gdb)
```

Здесь видно, что программа использует функцию div описанную в её коде

По результатам ответить на вопросы:

1. С какими ключами нужно скомпилировать программу, чтобы можно было пользоваться отладчиком gdb? Что произойдет, если собрать программу без этого ключа и загрузить её в gdb?

Для того, чтобы можно было использовать gdb, в исполняемый файл нужно добавить отладочную информацию с помощью ключа -g
Если ключ не добавить появится сообщение от gdb (No debugging symbols found in app.exe), и не работают команды gdb,
например, break (No symbol table is loaded)

2. Как запустить программу под отладчиком? Как досрочно завершить её работу?

Для запуска программы под отладчиком, нужно добавить в исполняемый файл отладочную информацию с помощью ключа -g
(gcc -Werror -g main.c -o app.exe) затем запустить gdb (gdb ./app.exe)
Что бы досрочно завершить работу используется команда quit

3. Выполнение программы было остановлено на какой-то из точек останова. Как понять, где именно остановилась программа?

gdb автоматически выводит строку, ее номер и номер точки останова в случае остановки.

4. С помощью какой команды можно посмотреть значение переменной? Изменить значение переменной?

Что бы посмотреть значения переменных используется команда info locals
Что бы изменить значение переменной используется команда set (set var [имя_переменной] = [значение_переменной])

5. С помощью каких команд можно выполнить программу в пошаговом режиме? Чем отличаются эти команды друг от друга?

С помощью команды next можно выполнять программу в пошаговом режиме, однако не осуществляются заходы внутрь функций
С помощью команды step можно выполнять программу в пошаговом режиме, с заходом в функции

6. Выполнение программы было остановлено на какой-то из точек останова. Как понять, какая последовательность вызовов функции привела сюда?

команда backtrace выводит цепочку вызовов стека

7. Как можно установить точку останова в программе?

Точку остановы можно установить с помощью команды break. Если после break ввести номер строки, то точка остановы будет поставлена на соответствующую строку, так же если после break написать имя функции, то точка остановы будет поставлена в начало этой функции.

8. Какая точка останова считается временной?

Такая точка остановы, которая удаляется после первой активации (tbreak)

9. Как сложно выключить/включить точку останова или пропустить некоторое количество срабатываний?

Что бы выключить/включить точку останова нужно использовать команду disable/enable [номер/диапазон].

Что бы пропустить некоторое количество срабатываний используется команда ignore номер количество итераций

10. Как задать условие остановки на точке останова?

Что бы задать условие остановки используется (break позиция if условие)

11. Чем отличаются точки наблюдения от точек останова?

точка наблюдения отличается от точки остановы, тем что точки наблюдения позволяют получать уведомления, когда происходят изменения в памяти. То есть точка наблюдения останавливает программу, когда меняются данные, а точка остановы, когда выполнение программы достигнет определенной строчки кода.

12. Приведите пример ситуации, в которой удобно использовать точку наблюдения.

Точку наблюдения удобно использовать во вложенных циклах или больших, например есть несколько циклов for, и некоторые переменные, которые меняются в этих циклах, нужно остановить программу, когда изменяемые переменные, например равны или отличаются друг от друга на некоторое число. Так же точки наблюдения удобно использовать при изменении переменных в программе через указатель и при рекурсии

13. С помощью какой команды можно посмотреть содержимое области памяти?

Что бы посмотреть содержимое области памяти используется команда x[/nfu] [адрес]

где n - сколько сколько единиц памяти должно быть выведено
f - спецификатор формата
u – размер выводимой единицы памяти

Задание №2

Составьте таблицу размеров типов char, int, unsigned, long long, short, int32_t, int64_t на двух разных машинах.

Тип	Linux	MacOS
Char	1	1
Int	4	4
Unsigned	4	4
Long long	8	8
Short	2	2
Int32_t	4	4
Int64_t	8	8

Задание №3

Показать, как в памяти представлены переменные типов char, int, unsigned, long long. Рассмотреть как положительные значения этих переменных, так и отрицательные. Результаты пояснить.

Код программы для выполнения

```c
#include <stdio.h>


int main(void)
{
    char cp = 6, cm = -6;
    int ip = 14, im = -14;
    unsigned up = 10, um = -10;
    long long llp = 1231.32, llm = -1231.32;
    return 0;
}
```

Таблица представления типов

Тип переменной	Положительное значение	Отрицательное значение
Char	(gdb) x /1xb &cp 0xfffffffffec7f: 0x06	(gdb) x /1xb &cm 0xfffffffffec7e: 0xfa
Int	(gdb) x /4xb &ip 0xfffffffffec78: 0x0e 0x00 0x00 0x00	(gdb) x /4xb &im 0xfffffffffec74: 0xf2 0xff 0xff 0xff
Unsigned	(gdb) x /4xb &up	(gdb) x /4xb &um

	0xfffffffffec70: 0x0a 0x00 0x00 0x00	0xfffffffffec6c: 0xf6 0xff 0xff 0xff
Long long	(gdb) x /8xb &llp 0xfffffffffec60: 0xcf 0x04 0x00 0x00 0x00 0x00 0x00 0x00	(gdb) x /8xb &llm 0xfffffffffec58: 0x31 0xfb 0xff 0xff 0xff 0xff 0xff 0xff

Представления отрицательных чисел отличается от положительных, так как отрицательные преобразованы в дополнительный код.

Задание №4

Показать, как в памяти представлен массив целых чисел.
Продемонстрировать особенности выполнения операции сложения указателя с целым числом на примере массива.

Код программы для выполнения задания:

```c
#include <stdio.h>


int main(void)
{
    int arr[] = {1, 2, 3, 4, 5};
    return 0;
}
```

Представление массива целых чисел в памяти:

```
(gdb) b 7
Breakpoint 2 at 0xaaaaaaaa07bc: file main.c, line 7.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/x13eav1sx/CLionProjects/test/a.out

Breakpoint 2, main () at main.c:7
7           return 0;

(gdb) x /20xb arr
0xfffffffffec68: 0x01    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0xfffffffffec70: 0x03    0x00    0x00    0x00    0x04    0x00    0x00    0x00
0xfffffffffec78: 0x05    0x00    0x00    0x00
```

Особенности выполнения операции сложения с целым числом на примере массива:

```
(gdb) p *arr
$1 = 1
(gdb) p *(arr+1)
$2 = 2
(gdb) p *(arr+3)
$3 = 4
(gdb) p *(arr+2)
$4 = 3
(gdb) p arr+2
$5 = (int *) 0xfffffffffec70
(gdb) p arr
$6 = {1, 2, 3, 4, 5}
(gdb) p arr+0
```

```
$7 = (int *) 0xffffffffec68
(gdb) p arr+5
$8 = (int *) 0xffffffffec7c
```

Можно заметить, что для числа, которое мы прибавляем к адресу первого элемента соответствует индексу соответствующего элемента, а для того, чтобы через gdb вывести указатель на первый элемент массива нужно прибавить к его адресу 0, иначе выводится сам массив.

Задание №5

Придумать пример программы с ошибкой, для нахождения которой удобно использовать точку наблюдения.

Код программы, демонстрирующий удобность использования точки наблюдения:

```c
// Пример программы в которой удобно использовать точку останова
#include <stdio.h>


int main(void)
{
    int a = 10;
    int b = 20;
    int *p = &a;
    a = b;
    b = *p;
    printf("a = %d, b = %d", a, b);
    return 0;
}
```

В данном случае удобно использовать точку наблюдения, чтобы следить за изменением значений переменных через указатель

```
(gdb) b 10
Breakpoint 1 at 0x7b4: file main.c, line 10.
(gdb) run
Starting program: /home/x13eav1sx/CLionProjects/test/a.out

Breakpoint 1, main () at main.c:10
10          a = b;
(gdb) watch a
Hardware watchpoint 2: a
(gdb) watch b
Hardware watchpoint 3: b
(gdb) c
Continuing.

Hardware watchpoint 2: a

Old value = 10
New value = 20
main () at main.c:11
11          b = *p;
```

```
(gdb) c
Continuing.

Watchpoint 2 deleted because the program has left the block in
which its expression is valid.

Watchpoint 3 deleted because the program has left the block in
which its expression is valid.
main () at main.c:17
17          printf("a = %d, b = %d", a, b);
(gdb) c
Continuing.
a = 10, b = 20[Inferior 1 (process 3401) exited normally]
```

Здесь видно, что значение b не меняется при выполнении программы, так как точка наблюдения за переменной b не сработала.

Задание №6

Изучить работу с отладчиком в Qt Creator. Написать инструкцию, которая сопоставляет команды gdb и аналогичные действия в Qt Creator.

Инструкция сопоставления команд gdb и действий в Qt Creator:

Gdb	Qt Creator
List	Окно редактора
Run	В нижнем левом углу есть кнопка
Break	Кликнуть мышкой левее от номера строки на которой нужно поставить точку останова
Info breakpoints	В окне «режимы отладки» есть информация о точках останова
Next	Есть кнопка в окне отладчика «перейти через»
Step	Есть кнопка в окне отладчика «войти в»