# Задание №3 в рамках вычислительного практикума

Отладка

## Задание 1.

### Программа 1:

Сначала я скомпилировал программу main_01.c для дальнейшей работы с отладчиком:

```
$ gcc -std=c99 -Wall -Werror -g main_01.c -o app_01.exe
```

Ключ -g нужен для работы с отладчиком. Он добавляет отладочную информацию в код. В ином случае отладчик gdb не сможет “шагать” по коду программы

Ключ -o нужен для создания исполняемого файла

Затем я проверил, что программа работает неправильно, Ввёл 5 и получил, ответ: factorial(5) = 0, что, очевидно, неверно, ведь factorial(5) = 120:

```
./app_01.exe
Input n: 5
factorial(5) = 0
```

После этого я перешел в режим отладки и посмотрел на код, с помощью команды list

```
$ gdb -silent ./app_01.exe
Reading symbols from ./app_01.exe...
(gdb) list 1,30
1       #include <stdio.h>
2
3       long long unsigned factorial(unsigned n);
4
5       int main(void)
6       {
7           unsigned n;
8           long long unsigned result;
9           printf("Input n: ");
10          if (scanf("%u", &n) != 1)
11          {
12              printf("Input error");
13              return 1;
14          }
15          result = factorial(n);
16          printf("factorial(%u) = %llu\n", n, result);
17
18          return 0;
```

```
19      }
20      long long unsigned factorial(unsigned n)
21      {
22          long long unsigned result = 1;
23          while (n--)
24              result *= n;
25          return result;
26      }
```

Так как вызов функции происходит на 15 строке, то я поставил точку остановы именно там с помощью команды “b 15”. Также я поставил точку остановы “b 24” на строке 24, так как именно там начинается алгоритм. После запуска отладки и попадания на 15 строку я ввёл команду “step” чтобы войти внутрь функции factorial(). Чтобы перейти ко второй точке останова я написал команду “continue”.

```
(gdb) b 15
Breakpoint 1 at 0x140001545: file main_01.c, line 15.
(gdb) b 23
Breakpoint 2 at 0x14000158c: file main_01.c, line 23.
(gdb) run
Starting program: D:\2 sem\ptp\app_01.exe
[New Thread 2316.0x288c]
Input n: 5

Thread 1 hit Breakpoint 1, main () at main_01.c:15
15          result = factorial(n);
(gdb) step
factorial (n=5) at main_01.c:22
22          long long unsigned result = 1;
(gdb) c
Continuing.
```

Чтобы узнать значения переменных result и n я использовал команду “p result” и “p n” соответственно. До захода в цикл result = 1, а n = 5. Затем командой “n” я зашел внутрь цикла и решил проверить значение переменной n. Оно стало равно 4. То есть result будет умножаться на 4, а не на 5, что уже является ошибкой.

```
Thread 1 hit Breakpoint 2, factorial (n=5) at main_01.c:23
23          while (n--)
(gdb) print n
$1 = 5
(gdb) print result
$2 = 1
(gdb) n
24              result *= n;
(gdb) print n
$4 = 4
```

Я решил дойти до последней итерации цикла, но делать это путем постоянных вводов “n” не хотелось, поэтому я сделал точку остановы с

условием “b 24 if n == 1” Сделав еще один шаг стало понятно, что цикл не завершается после того, как n стало равно 1, а идет дальше. После чего result умножается на n, который уже равен 0.

```
(gdb) list
19      }
20      long long unsigned factorial(unsigned n)
21      {
22          long long unsigned result = 1;
23          while (n--)
24              result *= n;
25          return result;
26      }
(gdb) b 24 if n == 1
Breakpoint 3 at 0x7ff75940158e: file main_01.c, line 24.
(gdb) c
Continuing.
[New Thread 2316.0x1a04]

Thread 1 hit Breakpoint 3, factorial (n=1) at main_01.c:24
24              result *= n;
(gdb) p n
$5 = 1
(gdb) p result
$6 = 24
(gdb) n
23          while (n--)
(gdb)
24              result *= n;
(gdb) p n
$7 = 0
(gdb) n
23          while (n--)
(gdb) p result
$9 = 0
(gdb) p n
$10 = 0
```

Таким образом ошибка работы программы стала очевидна. Так как в условии цикла есть инкремент “while (n--)”, то сначала n уменьшается и только потом умножается на result. Также этот цикл работает до тех пор, пока n > 0, но так как  n сначала уменьшится, став равным 0, и только на следующей итерации цикл завершится. В результате при вводе любого n > 0 программа всегда будет выводить 0.

Для исправления ошибки функцию стоит переписать следующим образом:

```
long long unsigned factorial(unsigned n)
{
    long long unsigned result = 1;
    while (n > 0)
    {
        result *= n;
        n--;
```

```
    return result;
}
```

**Программа 2:**

Таким же образом  скомпилировал программу task_02.и запустил app_02.exe файл в консоли:

```
$ ./app_02.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 5
Value [2] is 24
Value [3] is 0
Value [4] is -1331817576
The average is 0
The max is -1331817576
```

Видно, что программа работает неверно: Неправильный вывод массива, неправильное среднее и максимальное значения. Затем я запусти отладчик gdb, где сразу же вывел весь код программы.

```
$ gdb -silent ./app_02.exe
Reading symbols from ./app_02.exe...
(gdb) list 1,100
1       #include <stdio.h>
2
3       #define N 5
4
5       double get_average(const int a[], size_t n);
6
7       int get_max(const int *a, size_t n);
8
9       int main()
10      {
11          int arr[N];
12          size_t i;
13
14          printf("Enter %d numbers:\n", N);
15
16          for (i = 0; i < N; i++)
17          {
18              printf("Enter the next number: ");
19              if (scanf("%d", &arr[1]) != 1)
20              {
21                  printf("Input error");
22                  return 1;
23              }
24          }
25
```

```
26          for (i = 1; i < N; i++)
--Type <RET> for more, q to quit, c to continue without paging-
-REP
27              printf("Value [%zu] is %d\n", i, arr[i]);
28
29          printf("The average is %g\n", get_average(arr, N));
30
31          printf("The max is %d\n", get_max(arr, N));
32
33          return 0;
34      }
35
36      double get_average(const int a[], size_t n)
37      {
38          double temp = 0.0;
39
40          for (size_t i = 0; i > n; i++)
41              temp += a[i];
42          temp /= n;
43
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-
-RET
53                  max = a[i];
54
55          return max;
56      }
```

Затем я установил точку останова  на 19 строке, так как там начинается считывание элементов массива. После каждого введеного элемента я выводил весь массив и заметил, что изменяется только элемент массива под первым индексом. Происходит это из-за того, что в стркое ввода указано не “a[i]”, а “a[1]”. Поэтому все элементы записываются только в 1 индекс массива

```
(gdb) b 19
Breakpoint 1 at 0x140001532: file task_02.c, line 19.
(gdb) run
Starting program: D:\2 sem\ptp\app_02.exe
[New Thread 10712.0x1958]
Enter 5 numbers:
Enter the next number:
Thread 1 hit Breakpoint 1, main () at task_02.c:19
19              if (scanf("%d", &arr[1]) != 1)
(gdb) n
1
16          for (i = 0; i < N; i++)
```

```
(gdb) x/5d arr
0x5ffe80:       8       1       24      0
0x5ffe90:       11671000
(gdb) n
18              printf("Enter the next number: ");
(gdb)
Enter the next number:
Thread 1 hit Breakpoint 1, main () at task_02.c:19
19              if (scanf("%d", &arr[1]) != 1)
(gdb)
2
16          for (i = 0; i < N; i++)
(gdb) x/5d arr
0x5ffe80:       8       2       24      0
0x5ffe90:       11671000
(gdb) n
18              printf("Enter the next number: ");
(gdb)
Enter the next number:
Thread 1 hit Breakpoint 1, main () at task_02.c:19
19              if (scanf("%d", &arr[1]) != 1)
(gdb)
3
16          for (i = 0; i < N; i++)
(gdb) x/5d arr
0x5ffe80:       8       3       24      0
0x5ffe90:       11671000
(gdb)
```

После компиляции исправленной версии программы выводились не все элементы массива, но значения были верными. Среднее значение элементов массива и максимальный элемент массива по прежнему неверные

```
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
```

Поставил точку останова на 26 строку, так как там начинается вывод элементов массива и начал отладку:

```
(gdb) b 26
Breakpoint 1 at 0x7ff6bb15157d: file task_02.c, line 26.
(gdb) run
Starting program: D:\2 sem\ptp\app_02.exe
[New Thread 2520.0x2428]
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

Thread 1 hit Breakpoint 1, main () at task_02.c:26
26          for (i = 1; i < N; i++)
(gdb) n
27              printf("Value [%zu] is %d\n", i, arr[i]);
(gdb) info locals
arr = {1, 2, 3, 4, 5}
i = 1
```

На первой же итерации цикла переменная i равна 1, а не 0. Ошибка найдена.

После новой компиляции программы, массив выводился правильно, но среднее значение и максимальный элемент были по прежнему неверными:

```
$ ./app_02.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
```

В новой версии программы я поставил точки останова на 40 и 51 строках, так как там вычисляются среднее значение массива и максимальный элемент соответственно

```
(gdb) b 40
Breakpoint 1 at 0x14000162e: file task_02.c, line 40.
(gdb) b 51
Breakpoint 2 at 0x1400016db: file task_02.c, line 51.
(gdb) run
Starting program: D:\2 sem\ptp\app_02.exe
[New Thread 3816.0x3334]
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
```

```
Thread 1 hit Breakpoint 1, get_average (a=0x5ffe80, n=5) at
task_02.c:40
40          for (size_t i = 0; i > n; i++)
(gdb) n
42          temp /= n;
```

Видно, что программа не зашла в тело цикла, а с 40 строки перескочила на 42. Связано это с тем, что инициализированная переменная i=0, а аргумент n=5, в то время как условие работы цикла: i>n, что сразу выдает ложный ответ. Чтобы исправить нужно написать: i<n. Переходим к точке останова на строке 51:

```
51          for (size_t i = 1; i < n; i++)
(gdb) n
52              if (max > a[i])
(gdb) print max
$8 = 1
(gdb) print a[i]
$9 = 2
(gdb) n
51          for (size_t i = 1; i < n; i++)
(gdb) n
52              if (max > a[i])
(gdb) print max
$10 = 1
(gdb) print a[i]
$11 = 3
```

Значение a[i] всегда больше, чем значение max, но при этом никогда не происходит присваивание max=a[i]. Всё из-за того, что в условном операторе написано “if (`max > a[i]`)”. А должно быть “if (`max < a[i]`)”. Это нужно, чтобы значение max обновлялось, если текущий a[i] больше, чем max

После всех этих исправлений программа работает верно:

```
$ ./app_02.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 5
```

**Программа 3**

Таким же образом  скомпилировал программу task_03.и запустил app_03.exe в консоли:

```
$ gdb -silent ./app_03.exe
Reading symbols from ./app_03.exe...
(gdb) run
Starting program: D:\2 sem\ptp\app_03.exe
[New Thread 9144.0x28d8]
5 div 2 = 2

Thread 1 received signal SIGFPE, Arithmetic exception.
0x00007ff6bd23153e in div (a=10, b=0) at task_03.c:21
21          return a / b;
(gdb)
```

Консоль выводит информацию о том, что произошло исключение (арифметическая ошибка, деление на 0). Я вывел весь код программы и поставил точку останова на 21 строку. Так как там функция делит два числа и возвращает результат

```
(gdb) list 1,100
1       #include <stdio.h>
2
3       int div(int a, int b);
4
5       int main(void)
6       {
7           int a = 5, b = 2;
8
9           printf("%d div %d = %d\n", a, b, div(a, b));
10
11          a = 10;
12          b = 0;
13
14          printf("%d div %d = %d\n", a, b, div(a, b));
15
16          return 0;
17      }
18
19      int div(int a, int b)
20      {
21          return a / b;
22      }
(gdb) b 21
Breakpoint 1 at 0x7ff6bd23153a: file task_03.c, line 21.
(gdb) run
Thread 1 hit Breakpoint 1, div (a=5, b=2) at task_03.c:21
21          return a / b;
(gdb) n
22      }
(gdb)
5 div 2 = 2
main () at task_03.c:11
11          a = 10;
(gdb)
12          b = 0;
(gdb)
```

```
14          printf("%d div %d = %d\n", a, b, div(a, b));
(gdb) s

Thread 1 hit Breakpoint 1, div (a=10, b=0) at task_03.c:21
21          return a / b;
(gdb)

Thread 1 received signal SIGFPE, Arithmetic exception.
0x00007ff6bd23153e in div (a=10, b=0) at task_03.c:21
21          return a / b;
```

Словив то же самое исключение я вывел стэк вызова функций и обратился к нулевому фрэйму, так как именно в нем произошло исключение

```
(gdb) bt
#0  0x00007ff6bd23153e in div (a=10, b=0) at task_03.c:21
#1  0x00007ff6bd231506 in main () at task_03.c:14
(gdb) frame 0
#0  0x00007ff6bd23153e in div (a=10, b=0) at task_03.c:21
21          return a / b;
(gdb) info args
a = 10
b = 0
(gdb) info locals
No locals.
```

Посмотрев значения аргументов и локальных переменных стал понятно, что происходит деление на 0. a=10, b=0, функция возвращает a/b

**Ответы на вопросы:**

1) Чтобы можно было пользоваться отладчиком GDB, программу нужно скомпилировать с ключом -g, который включает информацию о отладке в исполняемый файл.
2) Чтобы запустить программу под отладчиком нужно использовать команду gdb ./название_файла. Если не хочется получать печать перед началом дебага, можно прописать gdb -silent ./нзвание_файла. Если нужно завершить выполнение программы, но остаться в отладчике, можно прописать kill. Если нужно завершить выполнение программы и выйти из отладчик, можно прописать quit или сокращенно q.
3) Чтобы узнать, в каком месте программы остановилось выполнение в отладчике GDB, можно использовать команду where или ее сокращенную версию bt (backtrace), также подойдет команда info frame или сокращенно i f.
4) Чтобы посмотреть значение переменной нужно использовать команду print или сокращенную версию p. Также можно и изменить значение переменной, например print p=2. Чтобы изменить значение переменной

нужно воспользоваться командой: set variable = новое_значение, например set x = 10

5) С помощью команд step и next. Step (step in) переходит к следующей строчке кода или заходит внутрь функций и продолжает свое действие там, в то время как next (step out) не заходит внутрь функций, а переходит к следующей строчке кода. При этом функции всё равно вычисляются
6) Использовать команду backtrace (bt). Она выведет стек вызовов функций, начиная от места текущей точки останова и возвращаясь к корню стека вызовов. Каждая строка вывода будет содержать информацию о функции, ее местоположении (имя файла и номер строки) и адресе возврата. Это помогает понять, какие функции и в каком порядке вызывались.
7) Команда break (b) устанавливает точку останова на заданной строке, функции или адресе программы
8) Временная точка останова - tbreak. Она автоматически удаляется после срабатывания. Способы задания те же, что и у точки останова break.
9) Командами disable, enable, ignore
    - disable – временно выключает точку останова
    - enable – временно включает точку останова
    - ignore – позволяет пропустить несколько срабатываний точки останова
10) Существуют условные точки останова, например: например: break 10 if n == 1. Если на 10 строке n будет равна 1, то выполнится точка останова
11) Точки останова (breakpoints) используются в отладчике для приостановки выполнения программы в определенной точке кода, что позволяет анализировать состояние программы в этом месте. Точки наблюдения (watchpoints) позволяют отслеживать изменения в значениях конкретных переменных в программе и автоматически приостанавливать выполнение программы при их изменении.
12) Отслеживание значения флага ошибки, отслеживание ввода пользователя, отслеживание процесса инициализации ошибки. Отслеживание переменных в цикле
13) С помощью команды **x/[количество]флаг[размер]**, где
    - [количество] - необязательный аргумент, указывающий количество элементов для отображения
    - флаг - указывает формат вывода данных. Например: b - байт (8 бит), d – десятичное число, x – шестнадцатиричное число, o – восьмиричное число и т.д.

- [размер] - необязательный аргумент, указывающий количество элементов, которые хотите просмотреть.

## Задание 2

Окружение: Windows 10, gcc 13.2.0

| Тип | Размер, байты |
|---|---|
| char | 1 |
| int | 4 |
| unsigned | 4 |
| short int | 2 |
| long int | 4 |
| long long | 8 |
| int_32t | 4 |
| int_64t | 8 |

Окружение: Ubuntu 22.04, gcc 11.4.0

| Тип | Размер, байты |
|---|---|
| char | 1 |
| int | 4 |
| unsigned | 4 |
| short int | 2 |
| long int | 8 |
| long long | 8 |
| int_32t | 4 |
| int_64t | 8 |

## Задание 3

| **Описание переменной** | **Представление в памяти** |
|---|---|
| char c1 = 'a' | 0x5ffe9f: 0x61 |
| char c2 = -100 | 0x5ffe9e: 0x9c |
| int i1 = 5 | 0x5ffe98: 0x05 0x00 0x00 0x00 |
| int i2 = -5 | 0x5ffe94: 0xfb 0xff 0xff 0xff |
| unsigned int ui1 = 10; | 0x5ffe90: 0x0a 0x00 0x00 0x00 |
| unsigned int ui2 = INT_MAX; | 0x5ffe8c: 0xff 0xff 0xff 0x7f |
| long long ll1 = 123456 | 0x5ffe98: 0x39 0x30 0x00 0x00 0x00 0x00 0x00 0x00 |
| long long ll2 = -123456 | 0x5ffe90: 0xc7 0xcf 0xff 0xff 0xff 0xff 0xff 0xff |

Стоит отметить, что память расположена в формате little-endian. Это формат, в котором младший байт числа хранится по меньшему адресу, а старший байт - по большему. Например, если имеется 4-байтовое целое число 0x12345678, то в little-endian формате оно будет храниться как 0x78 0x56 0x34 0x12.

char c1 = 'a':

- Адрес памяти: 0x5ffe9f
- Представление в памяти: 0x61
- Здесь переменная c1 хранит символ 'a'. В ASCII код символа 'a' равен 97 (0x61 в шестнадцатеричном формате), и этот байт будет представлен в памяти.

char c2 = -100:

- Адрес памяти: 0x5ffe9e
- Представление в памяти: 0x9c
- Переменная c2 хранит значение -100. В дополнительном коде -100 будет представлен как 0x9c.

int i1 = 5:

- Адрес памяти: 0x5ffe98
- Представление в памяти: 0x05 0x00 0x00 0x00
- Переменная i1 хранит значение 5, которое занимает 4 байта (размер типа int на вашей системе). Так как это значение положительное и младшие байты идут первыми (little-endian), то наименьший значащий байт (младший байт) равен 0x05.

int i2 = -5:

- Адрес памяти: 0x5ffe94
- Представление в памяти: 0xfb 0xff 0xff 0xff
- Переменная i2 хранит значение -5. В дополнительном коде -5 будет представлен как 0xfffffffb.

unsigned int ui1 = 10:

- Адрес памяти: 0x5ffe90
- Представление в памяти: 0x0a 0x00 0x00 0x00
- Переменная ui1 хранит значение 10, которое также занимает 4 байта.

unsigned int ui2 = INT_MAX:

- Адрес памяти: 0x5ffe8c
- Представление в памяти: 0xff 0xff 0xff 0x7f
- Переменная ui2 хранит максимально возможное значение для беззнакового int на моей системе, это 2147483647

long long ll1 = 12345:

- Адрес памяти: 0x5ffe98
- Представление в памяти: 0x39 0x30 0x00 0x00 0x00 0x00 0x00 0x00
- Переменная ll1 хранит значение 12345, которое занимает 8 байт.

long long ll2 = -12345

- Адрес памяти: 0x5ffe98
- Представление в памяти: 0xc7 0xcf 0xff 0xff 0xff 0xff 0xff 0xff
- Переменная ll2 хранит значение -12345, которое также занимает 8 байт

## Задание 4

```
$ gdb -silent ./a.exe
Reading symbols from ./a.exe...
(gdb) list
1       #include <stdio.h>
2
3       #define SIZE 5
4
5       int main(void)
7           int arr[SIZE] = {10, 20, 30, 40, 50};
8           int *ptr = arr;
9           return 0;
10      }
```

В этом коде я инициализировал целочисленный массив на 5 элементов и указателю ptr присвоил адрес первого элемента массива. Затем я вывел весь массив и 5 раз вывел указатель. Каждый следующий вывод я сдвигал его на 1 элемент дальше по массиву:

```
(gdb) x/20xb arr
0x5ffe80:       0x0a    0x00    0x00    0x00    0x14
0x00    0x00    0x00
0x5ffe88:       0x1e    0x00    0x00    0x00    0x28
0x00    0x00    0x00
0x5ffe90:       0x32    0x00    0x00    0x00
```

```
(gdb) x/4xb ptr
0x5ffe80:       0x0a    0x00    0x00    0x00
(gdb) x/4xb ptr + 1
0x5ffe84:       0x14    0x00    0x00    0x00
(gdb) x/4xb ptr + 2
0x5ffe88:       0x1e    0x00    0x00    0x00
(gdb) x/4xb ptr + 3
0x5ffe8c:       0x28    0x00    0x00    0x00
(gdb) x/4xb ptr + 4
0x5ffe90:       0x32    0x00    0x00    0x00
```

При выводе массива я написал x/20xb, так как на моей системt int занимает 4 байта, 4 * 5 = 20. Каждые 4 байта в выводе обозначают разные элементы массива. Так как каждый элемент массива занимает 4 байта, то и в указателе я выводил по 4 байта