Гайд о том, как пользоваться gdb – самый минимум для отладки 1-2-3 лабы и выполнения домашнего задания.

**Версия от 03.11.2022**

**Автор**: Штанов Евгений Юрьевич

Есть вопросы? Для связи со мной:

<https://t.me/Klavishnik0_o>

<https://vk.com/klavishnik0_o>

shtanov.klavishnik@yandex.ru

1. Что это такое?

GDB – это отладчик. Самая полезная для вас (в данный момент) его функция – способность «заморозить» состояние программы и выполнять её пошагово.

1. Как начать пользоваться

Первое действие – необходимо собрать вашу программу с отладочными символами. Это нужно для того, чтобы «под отладкой» вы видели исходный код вашей программки, а не ассемблерные команды.

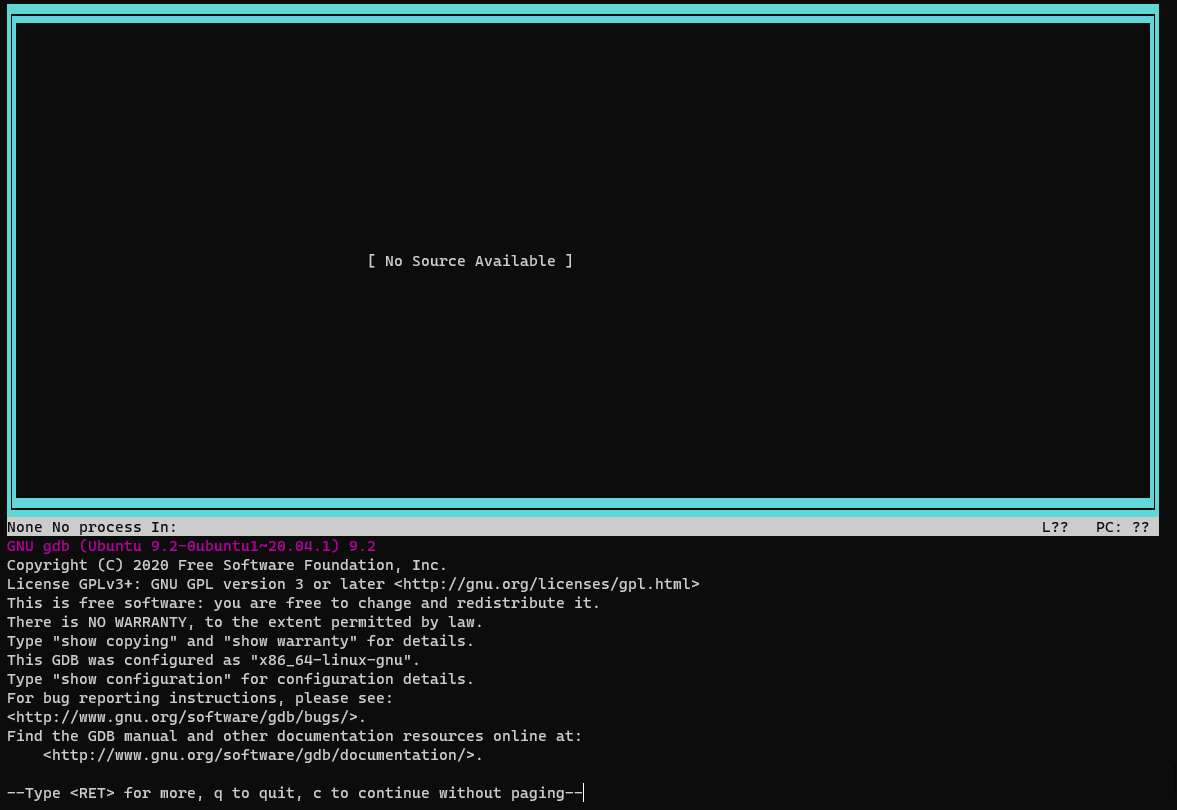
Пусть файл с исходным кодом у нас называется **1.с**, а скомпилированный бинарный файл будет иметь название **bin.** Флагом -g добавим отладочные символы к бинарному файлу. Для этого выполним следующие команды:

gcc 1.c -o bin -g

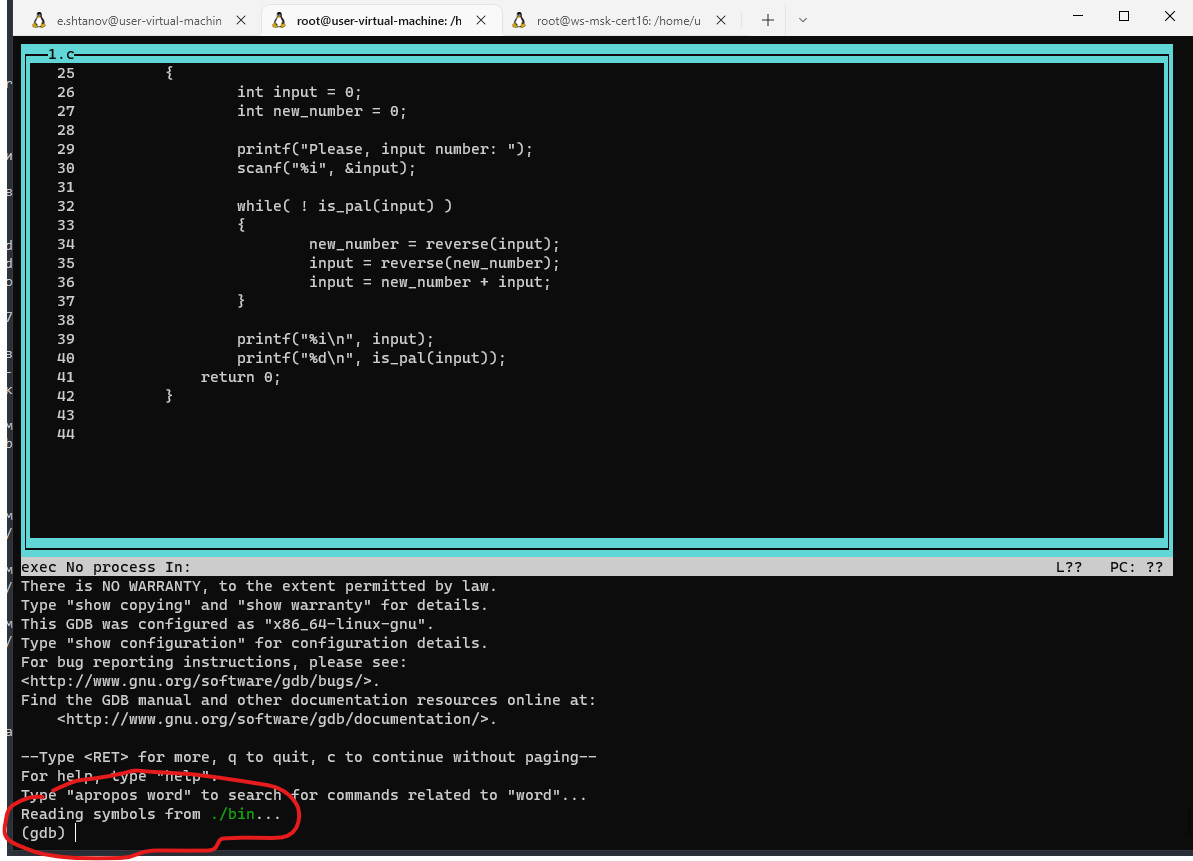
А теперь запустим отладчик. Запускается он командой **gdb** и на вход принимает название бинарного файла **bin**. Флаг **-tui** нужен для запуска в интерактивном режиме. Команда:

gdb -tui ./bin

1. Стадия отрицания

После запуска вы увидите такое окно  


Нажимаем клавишу **Enter**



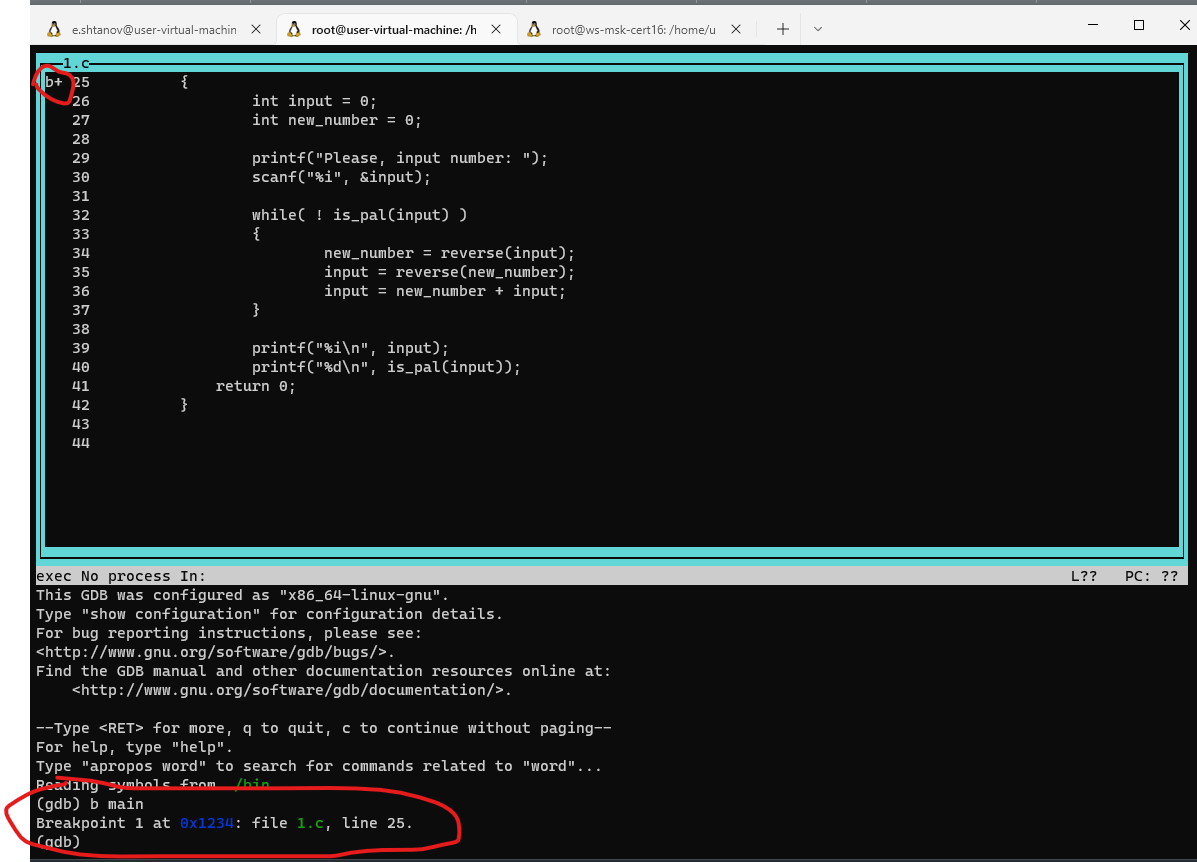
Видно, что появился доступ работе с окружением gdb и отладчик подгрузил отладочные символы для bin, поэтому в окне выше показан исходный код.

1. Стадия гнева

Для того, чтобы начать пользоваться отладчиком, нужно запустить программу. НО, если мы ее запустим сейчас, то она выполнится вся. Нужно остановить выполнение программы. Для этого существует механизм **«точек останова»** или **breakpoint**. Если коротко – это своеобразная «метка», которая ставится на строку, при достижении которой отладчик остановит выполнение вашей программы, зафиксировав её состояние.

Чтобы поставить точку останова необходимо знать название функции или номер строки. Поскольку точка входа в нашу программу является функция **main** (т.е. для нас это начало нашей программы), то на неё и поставим брейкпоинт командой:

b main

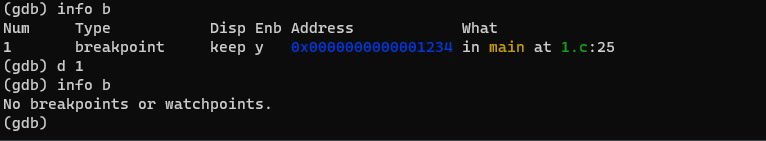
Видно, что появился указатель с точкой останова **b+**, а в выводе дана информация на какой строке стоит точка останова   


Чтобы посмотреть все точки останова нужно выполнить команду

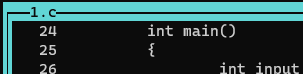
info b

Чтобы удалить точку останова нужно выполнить команду

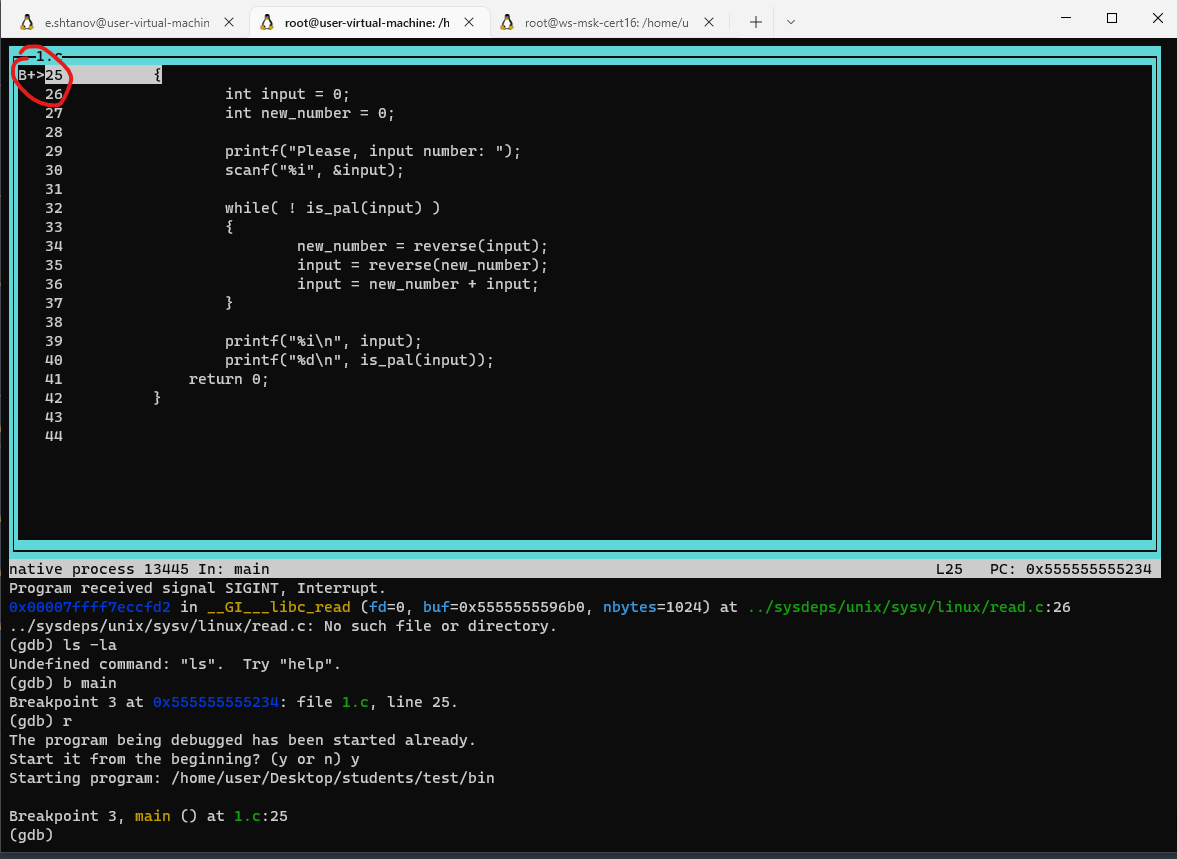
d 1, где 1 – это номер точки останова



Указатель на точку останова в области исходного кода тоже пропал



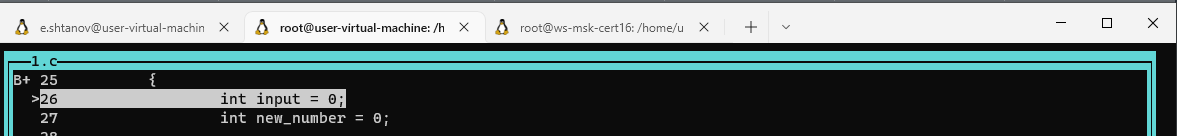
1. Стадия торга

Теперь начнем выполнять нашу программу. Установив точку останова, наберем **r (**от run – запуск, хотя можно и run написать**)**

Видно, что рядом с значком b+ появился значок **> -** это указатель на то, на какой строке мы находимся.

**В отладчике можно перемещаться только вперед – к предыдущему состоянию вернуться нельзя!**

Теперь попробуем пойти дальше. Для этого введем **n (от next)**



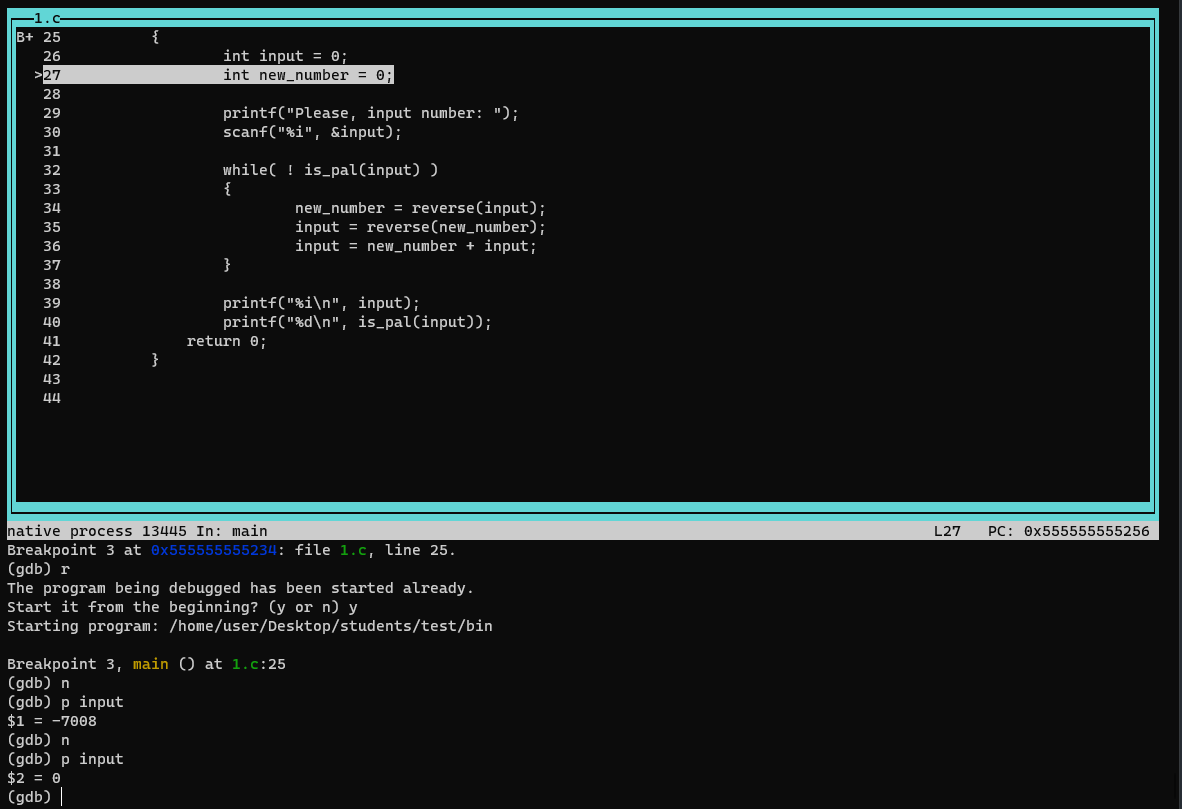
Видно, что перешли на новую строку, после выполнения которой будет инициализирована и объявлена переменная input.

Для того, чтобы посмотреть состояние переменных, используем команду **p название\_переменной (от print)**

**p input**

****

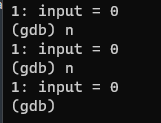
Переменная имеет такое значение, поскольку еще не проинициализирована новым значением и хранит в себе данные, оставшиеся от прошлых использований.

Сделаем **n и print input еще раз**

Наглядно видно, что теперь переменная имеет нулевое значение.

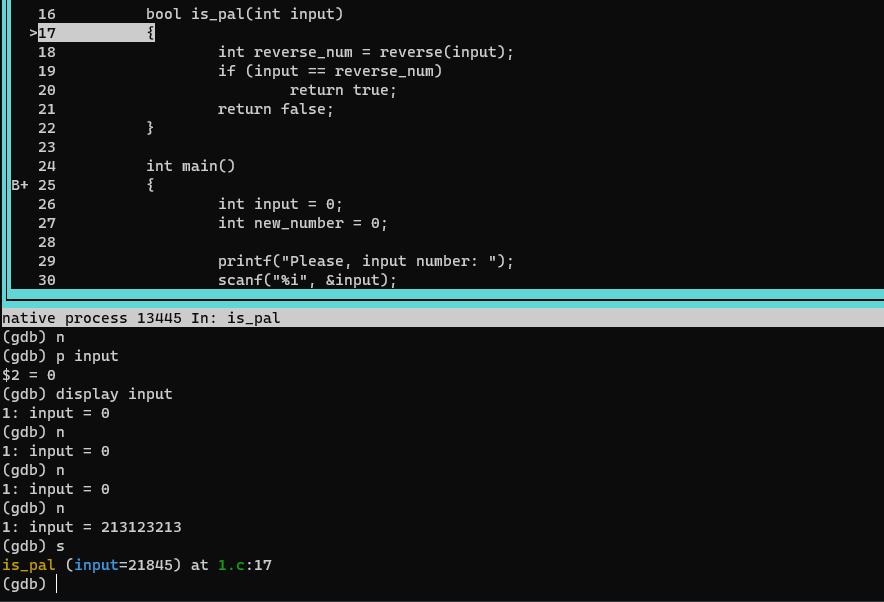
Чтобы каждый раз не писать **print input** можно использовать конструкцию **display название\_переменной** - тогда содержимое переменное будет **выводиться на каждом шаге**.

**display input**



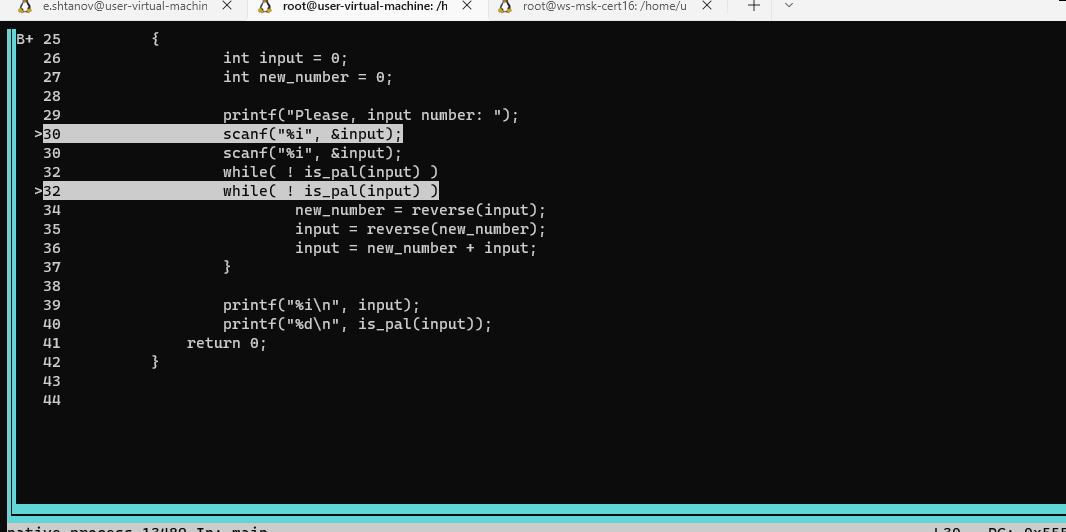
1. Стадия депрессии

Конструкциями next нельзя зайти внутрь функции – шаги будут идти строго по порядку в main. Для захода внутрь функции необходимо дойти до ее вызова и **оказавшись на ней** выполнить **s.**



1. Стадия принятия

Если у вас «поехал» экран, как тут:



То нажмите комбинацию **ctrl+l**

Чтобы выйти из gdb нажмите **q**

Чтобы посмотреть все переменные в области видимости выполните **info locals**



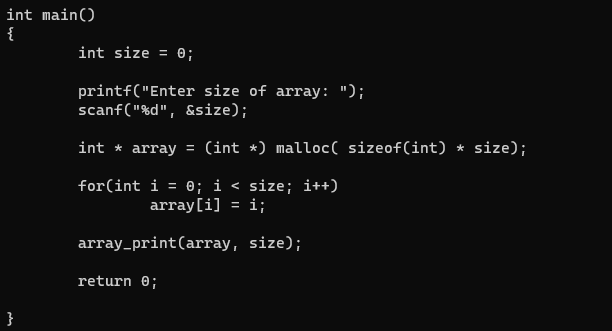
1. Массивы

А что делать с массивами?

Далее идут сакральные знания.

1. Для начала нужно вспомнить, переменная в памяти - это одна ячейка. А массив данных в памяти - это набор таких ячеек, которые расположены друг за другом. Напоминаю, что в Си работа с массивами организована через указатель - переменную, **значение** которой - ЭТО АДРЕС ПЕРВОГО ЭЛЕМЕНТА МАССИВА.
2. GDB по синтаксису и возможностям очень похож на Си.

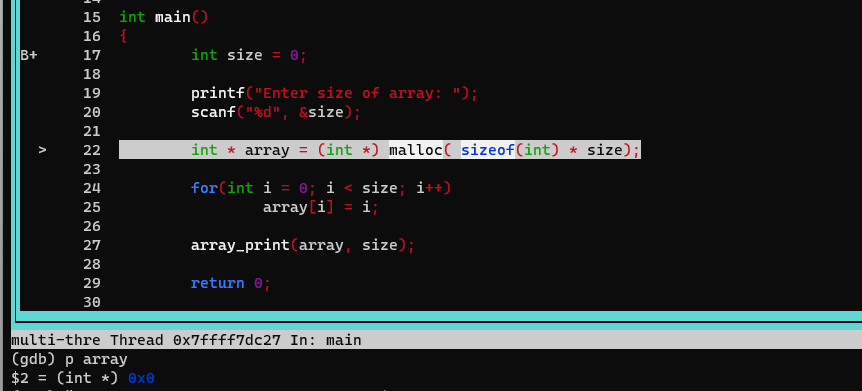
А теперь перейдем к экспериментам:



Есть вот такой код. Видно, что тут вводится с клавиатуры количество элементов массива, дальше под массив выделяется динамически память, а потом каждому элементу массива приравнивается значение его индекса.

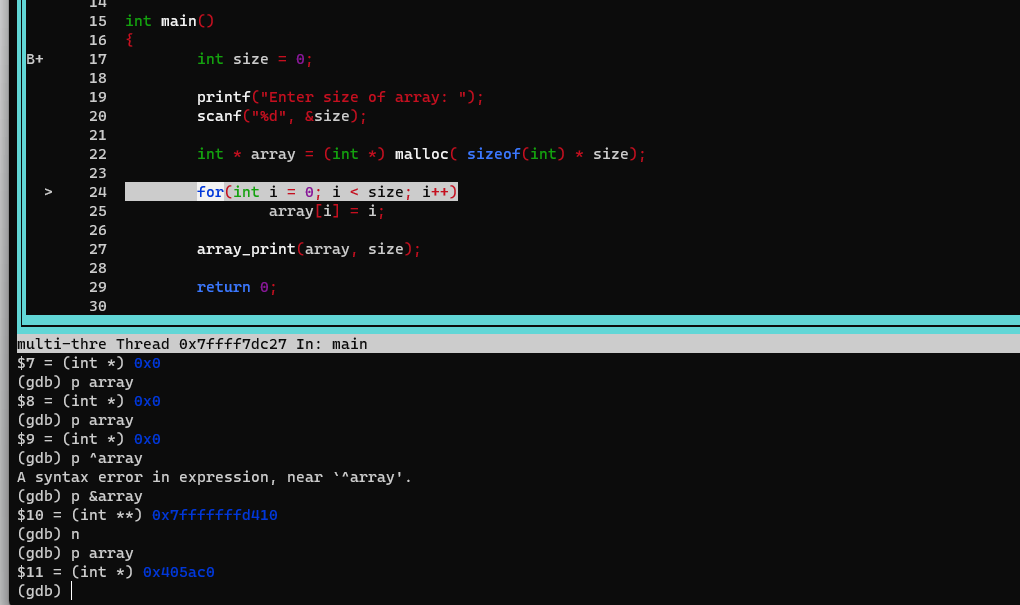
С этим разобрались.

Если раньше переменную мы смотрели командой p (print), то и массив можно посмотреть этой командой. Логично? Логично. Смотрим.



Почему ноль? Потому что в данный момент массив не инициализирован! ПАМЯТЬ ЕЩЕ НЕ ВЫДЕЛЕНА. УКАЗАТЕЛЬ УКАЗЫВАЕТ В ПУСТОТУ.

Делаем **n** и **p array** смотрим, что получилось



Видим, что теперь в указателе **array** хранится какой-то адрес.

А этот адрес - это место, где будут расположены данные нашего массива.

Указатель содержит адрес, по которому хранятся данные (повторяю 100500 раз потому что вы не понимаете).

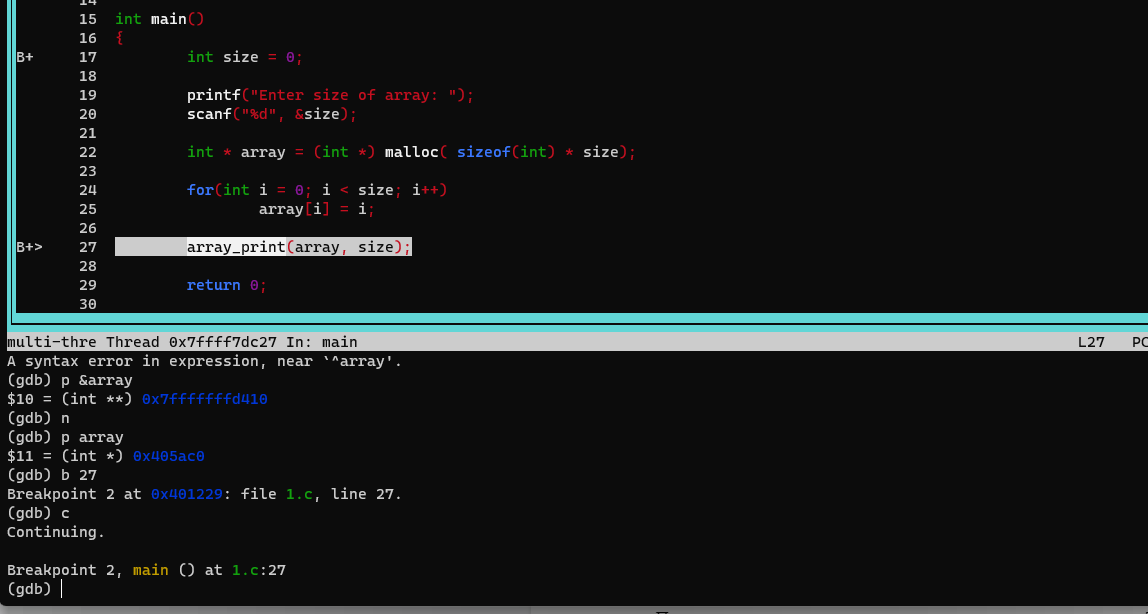
Если мы сейчас залезем в этот участок памяти, то мы там ничего нормального не увидим - элементы массива не проинициализированы, поэтому там будет валяться “мусор” - данные от операционной системы.

“Перескочим” участок кода, где происходит инициализация массива.

Кидаем точку останова на строку 27 и продолжаем выполнение программы:

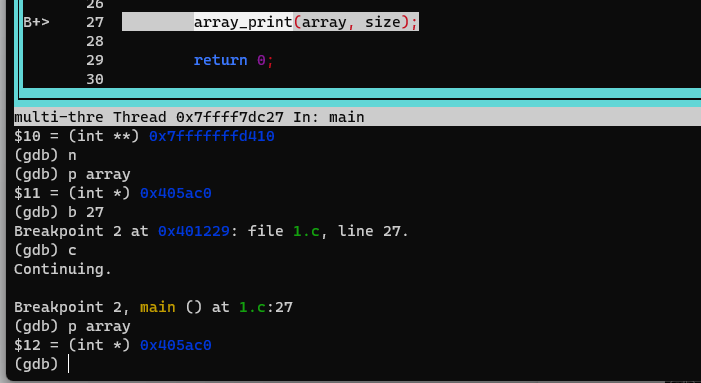
**b 27**

**c**



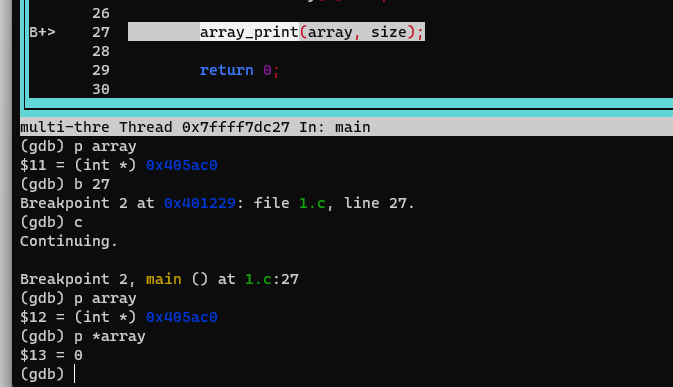
Смотрим, что у нас хранится в **array**

p array



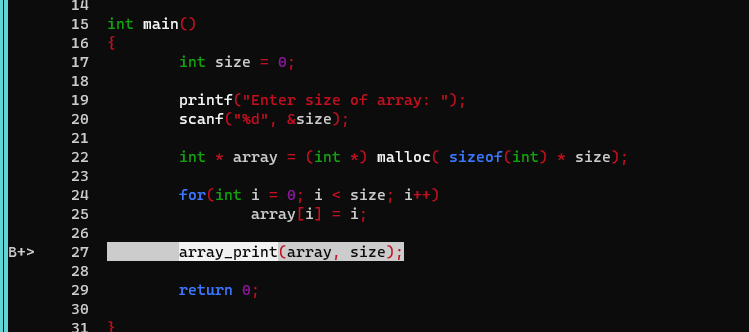
Хм, ничего не изменилось, как же так.   
А все правильно! Мы смотрим то, чему равно значение указателя! А там адрес! И он не менялся! А изменилось значение по этому адресу!  
А как посмотреть?  
Да как и в Си, с помощью операции разыменования

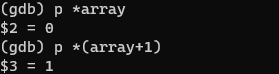
p \*array



Ноль! Правильно, это ведь значение нулевого элемента массива.

А давайте посмотрим чему равен следующий элемент?  
p \*(array+1)





Единица. Все правильно.

Таким способом можно хоть весь массив посмотреть. Например третий элемент  
p \*(array+3)



Вроде все верно.

Т.е. конструкция имеет вид  
**p \*(name\_array + i), где i - номер индекса массива**

А как чекнуть то весь массив?  
Да как и в Сишке! Проблема фундаментальная - нужно знать какой длины у тебя массив.

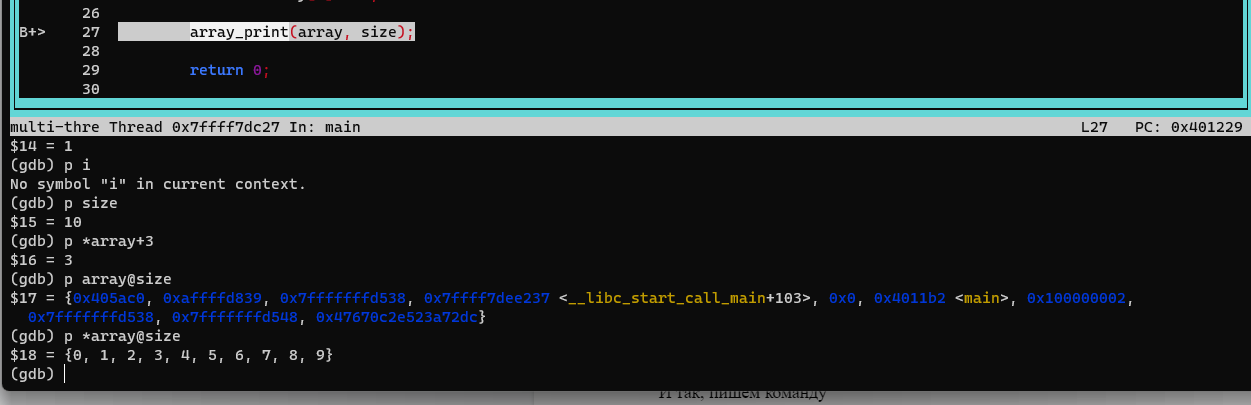
Конструкция выглядит так:  
**p \*name\_array@size\_to\_display**

Причем здесь **size\_to\_display - это сколько элементов массива вывести**

Т.е. у массив длиной 10 элементов можно вывести на экран как 2 элемента, так и 14, только во втором случае вы залезете в “чужую” память на 4 элемента.

И так, пишем команду  
p \*array@size

(size - моя переменная у меня в коде объявленная, это размер массива)



(На картинке наглядно видно, что будет, если забыть поставить звездочку)  
Ну собственно вывелся весь массив.   
А если хочу посмотреть первые пять элементов?  
Пишем

p \*array@5



Работает.

1. Доступ к памяти

Ну в целом понятно. А как именно в ПАМЯТИ то данные посмотреть?

Есть такая конструкция

**x/NUMспецификатор\_системы\_счисленияразмер\_данных адрес**

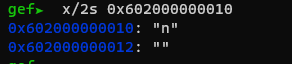
**(ПРОБЕЛ ЗДЕСЬ ОДИН!)**

Расшифровываю

**x/**- это команда вызова hex редактора gdb;

**спецификатор\_системы\_счисления -** в каком виде будет вывод на экран. Если написать **x**  то шестнадцатеричный, если **d** то десятичный**, с -** символ**;**

**ДЛЯ ОТДЕЛЬНОГО ВЫВОДА СТРОКИ ИСПОЛЬЗУЙ СПЕЦИФИКАТОР S**

****

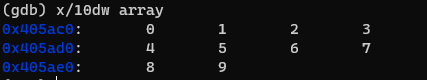
**размер\_данных** - **b** - байт, **h** - половина слова (2 байта) **w** - слово (4 байта) **g** - двойное слово (8 байт)

**NUM - сколько элементов выбранной размерности показать**

**адрес -** говорит само за себя. Тут в качестве аргумента передаем адрес (хоть ручками набранный 0х777ff…., хоть переменную-указатель)

Например, хочу вывести на экран все 10 элементов моего массива (10 элементов типа данных int (4 байта) в десятичном виде):

**x/10dw array**

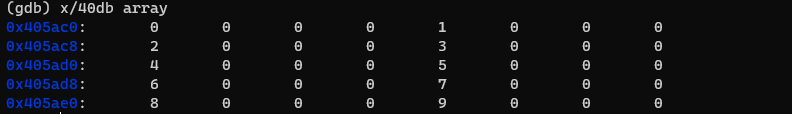


Слева видно адреса. Первый адрес - это адрес, который хранится в указателе (ему соответствует значение 0). Единица будет храниться по адресу 0x105ac0 + 4 (размер int) или (как мы пишем в Си) \*(array+1).

Между строками разница в 0x10 или в 16 элементов. Т.е. на одной строке у нас вмещается 4 элемента по 4 байта.

А теперь хочу вывести на экран все элементы моего массива по байтово в десятичном виде (40 элементов типа данных byte (1 байт \* 10 элементов \* 4 байта)):

x/40db array



Вывод побайтовый. Именно так хранится информация у вас в кампухтере.

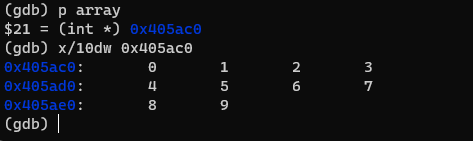
Помните! Под разные типы данных выделяется разное количество памяти. Так тип данных char или byte занимает в памяти 1 байт, а тип данных int - 4 байта. (Конечно все еще зависит от архитектуры машины бла бла …, но у нас именно так).

Поэтому мы можем смотреть разные данные по-разному - int как байт, как было сделано выше.

А если я хочу посмотреть какой-то определенный адрес?   
Да пожалуйста.  
Сначала получим адрес нашего массива, а потом глянем его в памяти ручками

p array

**x/10dw адрес**

****

Все то же самое, что и через array.

1. Представление чисел в кампухтере

Тут должна быть отдельная глава о том, как представляются числа в компьютере.   
Честно - мне лень делать чужую работу, поэтому пишу коротко, остальное есть в гугле.

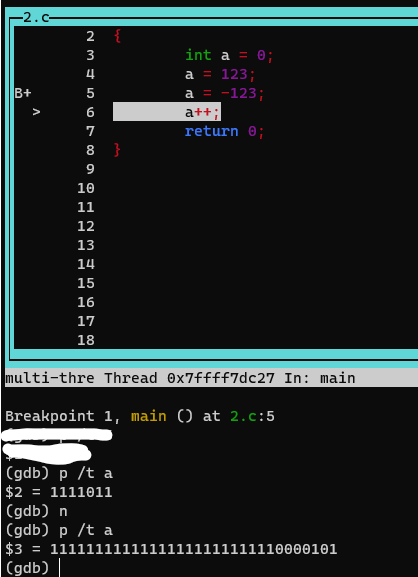
Есть три типа представления числа - прямой, обратный и дополнительные коды.  
Положительные числа не представляют проблем, поэтому хранятся в прямом коде (это их чисто бинарый вид, можете в кУлькУляторе винды перевести число из десятичного в двоичное и получите его в прямом коде).

Дополнительный и обратный используются для представления отрицательных чисел.

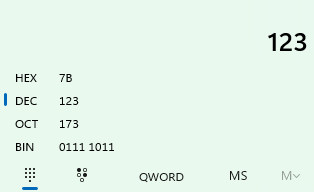
А как в компьютере то хранится это все дело? Как посмотреть?  
Все очень просто.   
Пишем код, запускаем отладчик.  
И делаем такую команду

**p /t название переменной**

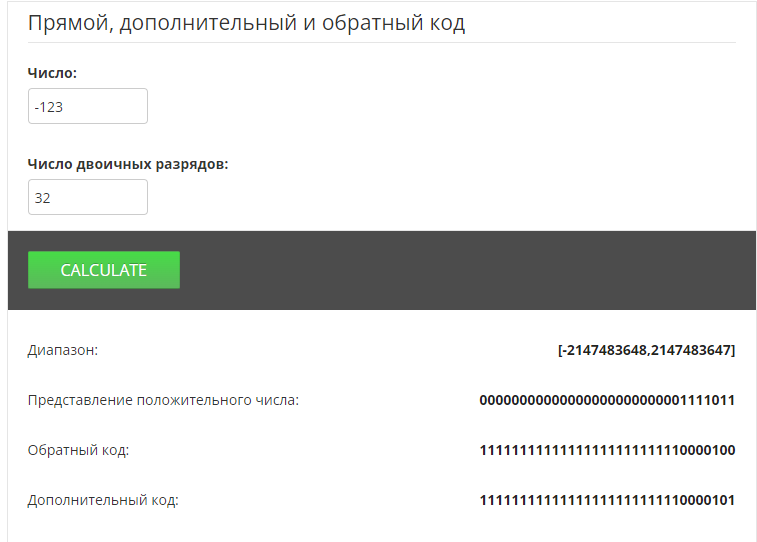
Смотрим



Видно, что число **123** отобразилось в прямом коде



А число **-123** в дополнительном





От автора:

*Пока я гуглю за вас инфу, моя ценность на рабочем рынке поднимается все выше и выше.*

*Учитесь делать это самостоятельно!   
Не нашли ответ в ру сегменте сети? Переведите ваш запрос на английский, зайдите в браузер в режиме инкогнито и выполните поисковый запрос в гугле. Большая часть ответов будет на англоязычном сегменте сайта StackOverflow. Информацию там можно перевести с помощью встроенного в браузер переводчика.*

[Примеры работы санитайзеров ASAN и MSAN на типовых ошибках, возникающих при работе массивами.](https://github.com/Klavishnik/sanitizers-examples) (<---- Это ссылка на репозиторий, нажми на неё)  
  
Как собрать проект, состоящий из нескольких файлов? Как скомпилировать статическую или динамическую библиотеку? Смотри [здесь](https://github.com/Klavishnik/How-to-compile-project).