Отладка в CLion

Для примера рассмотрим такую программу:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <fcntl.h>

6  #define N 10

8  int main()
9  {
10     unsigned long seed;
11     read(open("/dev/random", O_RDONLY), &seed, sizeof(seed));
12     srand(seed);

14     int a[N], i, s = 0;
15     printf("My array:");
16     for (i = 0; i < N; ++i)
17     {
18         a[i] = rand() % N;
19         s += a[i];
20         printf("%2d",  a[i]);
21     }
22     printf("\nSumm: %d\n", s);
23     return 0;
24 }
```

Несмотря на то, что программа является заведомо рабочей, на ней можно продемонстрировать некоторые моменты отладки.

Чтобы начать процесс отладки, сначала нужно поставить точку останова на какой-либо строке кода. Это можно сделать как минимум тремя способами:

- кликнув по пустой области между номером нужной строки и вертикальной чертой, правее которой мы пишем код (см рис. 1);
- командой `Toggle line breakpoint` из меню `Run`;
- нажав `Ctrl+F8`.

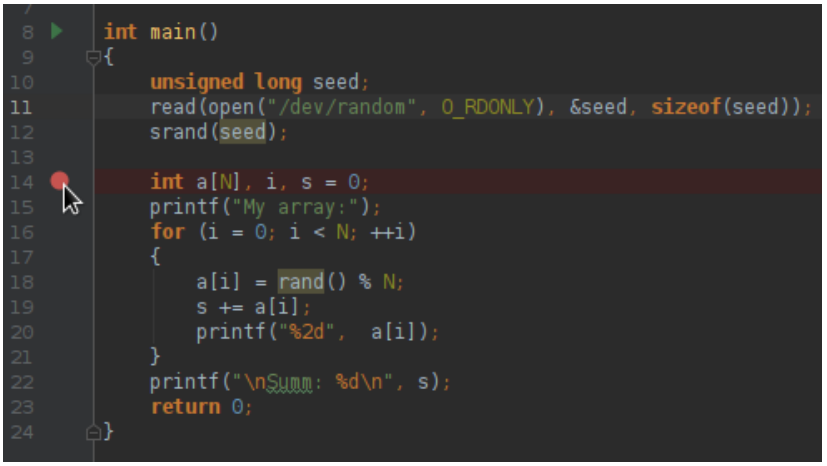

Рис. 1: Точка останова на строке 14

Два последних способа корректно сработают при условии, что текстовый курсор находится на нужной строке кода. Можно также поставить временную точку останова (`Ctrl+Alt+Shift+F8`), она отличается от обычной тем, что пропадёт после первого срабатывания.

Если всё сделано верно, рядом с номером строки появится красный круг — маркер точки останова (рис. 1).

Теперь для начала процесса отладки нужно нажать кнопку с изображением зелёного жука в правой верхней части окна CLion (или `Shift+F9`). Проект будет пересобран и запущен в режиме отладки. В примере точка останова находится на 14 строке, это означает, что программа будет остановлена на моменте **перед** выполнением этой строки, т.е. при срабатывании этой точки останова весь код до 14 строки программа уже отработала. После срабатывания точки останова над красным кругом появится галочка, означающая, что эта точка останова активна, т.е. будет срабатывать.

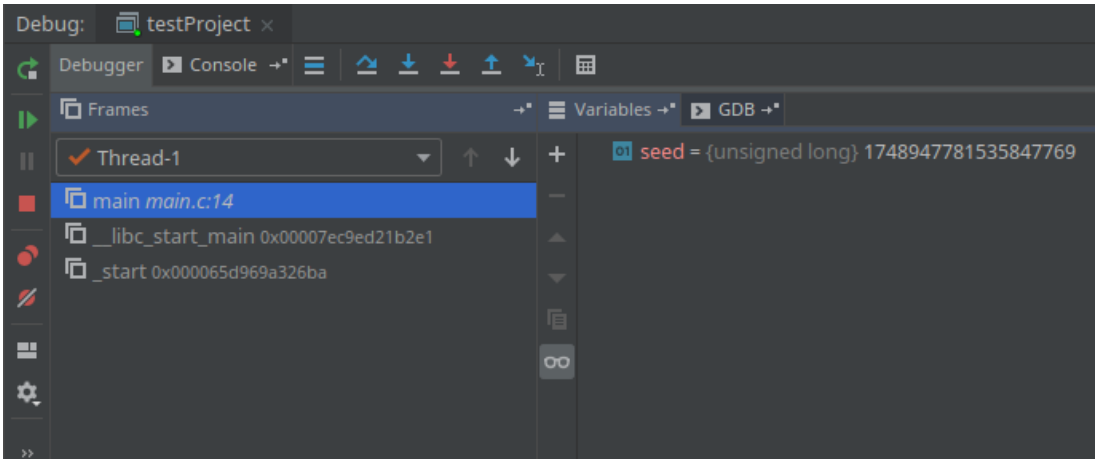

Рис. 2: Палитра `Debug`

В нижней части окна CLion будет открыта палитра `Debug` (рис. 2) на вкладке `Debugger`. В палитре есть вертикальный ряд кнопок слева, содержащий кнопки управления запуском программы, точками останова и две служебные кнопки — кнопку восстановления палитры `Debug` к виду по умолчанию, а также кнопку настроек. Горизонтальный ряд кнопок сверху содержит кнопки управления ходом и шагами отладки. На вкладке `Debugger` расположены окно `Frames` и две вкладки `Variables` и `GDB`, отделённые от окна `Frames` вертикальным рядом кнопок управления просматриваемыми переменными (далее, для краткости будем называть просмотриваемую переменную watch).

В окне `Frames` находится выпадающий список потоков нашей запущенной программы (в примере единственный поток), а также список кадров стека вызовов. В примере мы находимся в функции `main()`, поэтому этот элемент списка активен. При нажатии на другие элементы мы “проваливаемся” в дизассемблированную программу и можем пошагово следить за ходом выполнения кода ассемблера.

На вкладке `Variables` доступен список переменных, видимых в данный момент выполнения кода. В примере, т.к. 14 строка ещё не выполнена, мы пока можем видеть только переменную `seed` и её значение (рис. 2).

Тем не менее, всегда можно добавить на вкладку `Variables` и собственные выражения (кнопка "+" или клавиша `Insert`), и, если значение выражения может быть вычислено на текущий момент — увидеть значение. Для примера добавим три watch: `N`, `i == N` и `k`. Получим значения только первых двух выражений, т.к. наша программа "не знает" что такое `k` — такая переменная нигде не объявлена (рис. 3).

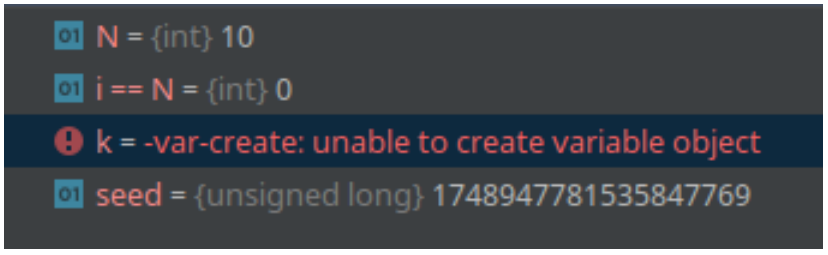

Рис. 3: Локальные переменныые и пользовательские watch

Чтобы сделать шаг по коду (т.е. выполнить одну строку), нужно нажать клавишу `F8` или выполнить команду `Step Over` из меню `Run`, либо нажать соответствующую кнопку на палитре `Debug`. Рекомендую изучить остальные варианты шагов по коду, их значение и способы применения.

Теперь в нашем примере выделена синим строка 15, соответственно, строка 14 уже выполнена и на вкладке `Variables` мы видим значения новых переменных. Как видно из рис. 4, массив `a` заполнен произвольными значениями, т.к. объявлен локально.

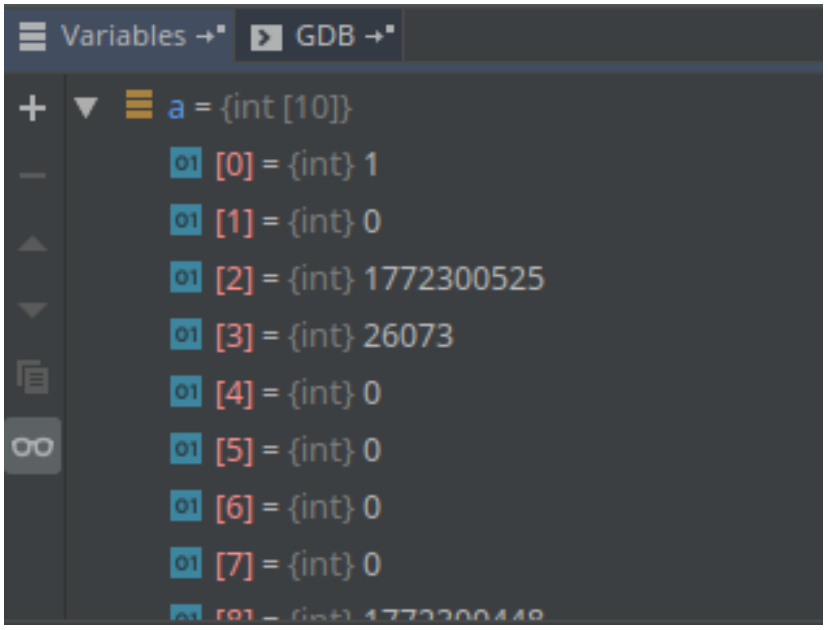

Рис. 4: Значения локального массива `a`

Продолжая выполнять шаги, зайдём в цикл `for` и будем двигаться по нему итерация за итерацией, пока не выполнятся все 10 — в это время можно наблюдать изменение элементов массива `a` после соответствующих присваиваний.

Не всегда удобно выполнять код циклов пошагово, дожидаясь появления ошибки, а зачастую хотя бы примерно известно, когда возникает ошибка. Для удоб-

ства отладки в таких случаях используются условные точки останова. Создать такую точку можно следующим образом: создать обычную точку останова, после этого нажать правой кнопкой мыши на маркер точки останова и в появляющемся окна ввести условие в поле `Condition`. Кроме того, в этом окне можно включить или выключить точку останова (`Enabled`), указать будет ли программа остановлена при достижении точки или нет (`Suspend`), и настроить другие параметры. По ссылке `More` откроется окно `Breakpoints` (его можно вызвать сочетанием `Ctrl+Shift+F8`, либо нажатием кнопки с двумя красными кружками на палитре `Debug`, либо соответсвующим пунктом меню `Run`), в котором присутвуют все инструменты по управлению точками останова, рекомендую их изучить.

Рис. 5: Условная точка останова

Для примера поставим точку останова на 19 строке, в поле `Condition` введём “`i == 7`” и запустим отладку. Программа будет остановлена при выполнении этого условия.

Предлагаю попробовать отладку самостоятельно на реальных программах.