

# Искусство отладки

Денис Адамчук



# Структура лекции

## Основные разделы и подразделы.

- ❑ Введение
- ❑ Отладка в контексте разработки
  - Неизбежность отладки
  - Процесс разработки
  - Тестирование
  - Информационное и программное обеспечение
- ❑ Инструменты отладки ПО
  - Отладчик
  - Профилировщик
  - Детектор утечек памяти
  - Анализаторы кода
- ❑ Отладчик Visual Studio
  - Точки останова
  - Выполнение программы
  - Стек вызовов
  - Просмотр переменных
  - Debug и Release
- ❑ Дампы памяти
- ❑ Типовые ошибки
- ❑ Q&A
- ❑ Литература

# Введение

**Отладка** – это поиск причин (или изучение) и устранение ошибок в программе.

“ Отлаживать код вдвое сложнее, чем писать.  
Если Вы используете весь свой интеллект  
при написании программы, вы по определению  
недостаточно умны, чтобы её отладить. ”



**Брайан Керниган**



# Отладка в контексте разработки

- Неизбежность отладки
- Отладка и процесс разработки
- Алгоритм поиска и устранения ошибки
- Информационное и программное обеспечение



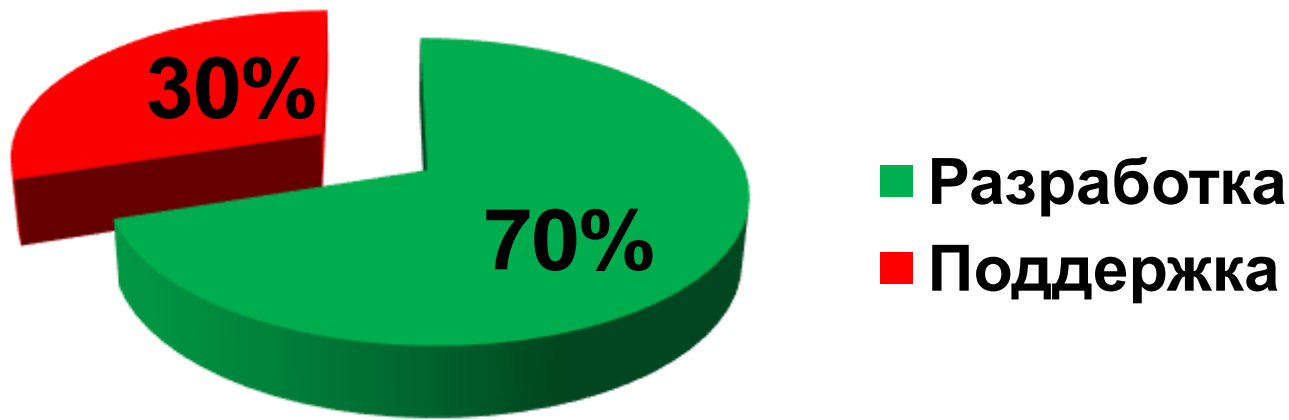
# Отладка в контексте разработки : Неизбежность отладки

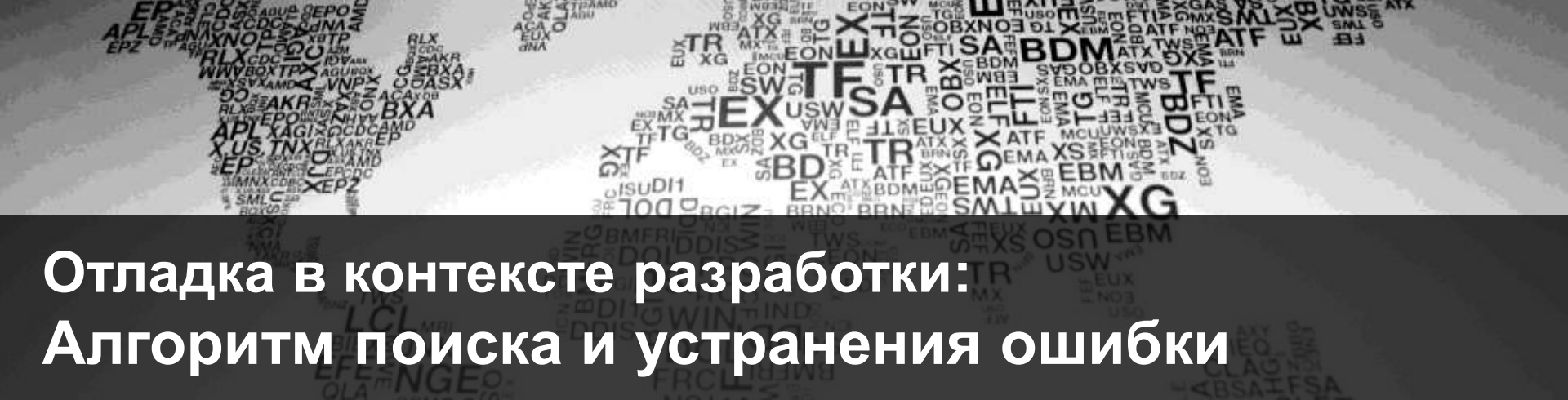
- Человеческий фактор
- Недостаток знаний и навыков
- Компромисс между качеством и сроками
- Неясные и меняющиеся требования



# Отладка в контексте разработки: Процесс разработки

В среднем 30% времени разработчиков тратится на поиск и исправление ошибок в существующем ПО





# Отладка в контексте разработки: Алгоритм поиска и устранения ошибки

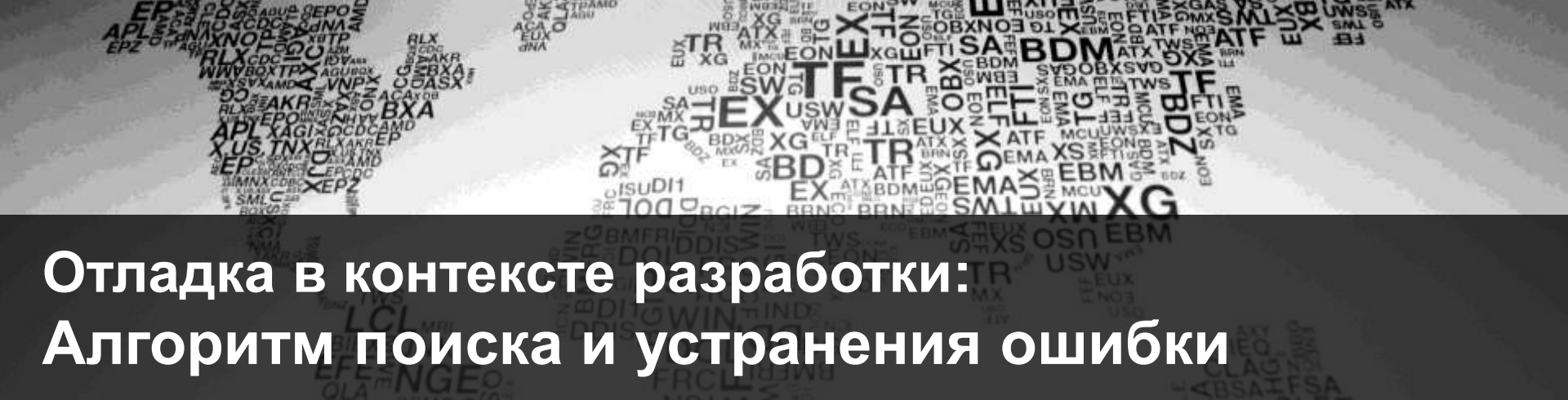
## Основные шаги:

Воспроизведение (reproducing)

Исследование (investigation)

Исправление (fixing)

Проверка (testing)

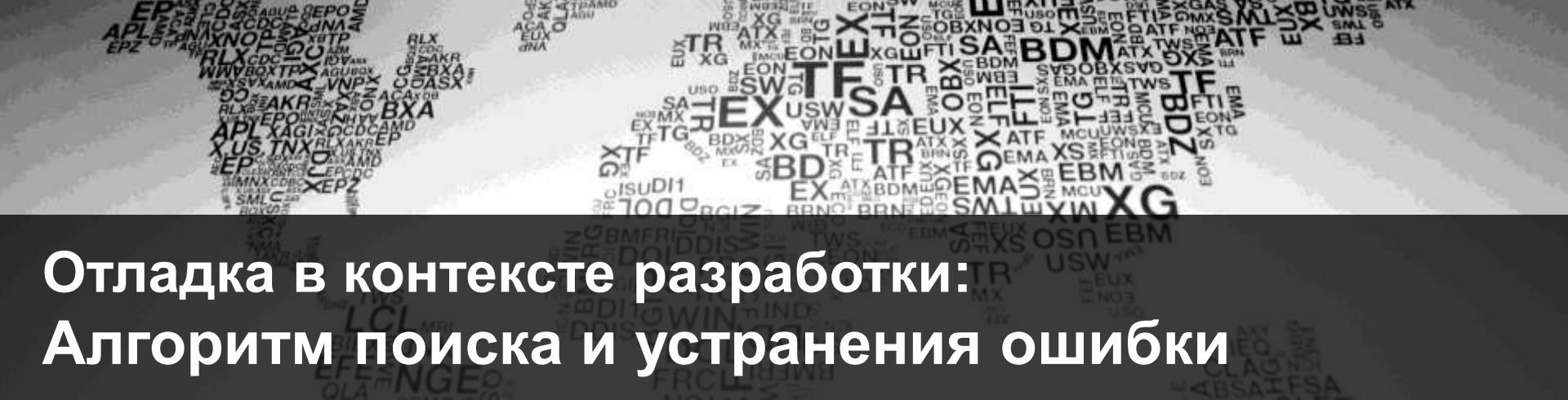


# Отладка в контексте разработки: Алгоритм поиска и устранения ошибки

## Основные шаги. Воспроизведение.

- Воспроизведенная проблема – 50% успеха
- Воспроизведение может быть долгим
- Что помогает воспроизвести проблему:
  - Инструкции или подсказки от пользователя
  - Код
  - Логи
  - Конфигурация
  - Отслеживание изменений в коде
  - Настойчивость и изобретательность
- Формальный результат: шаги или ничего

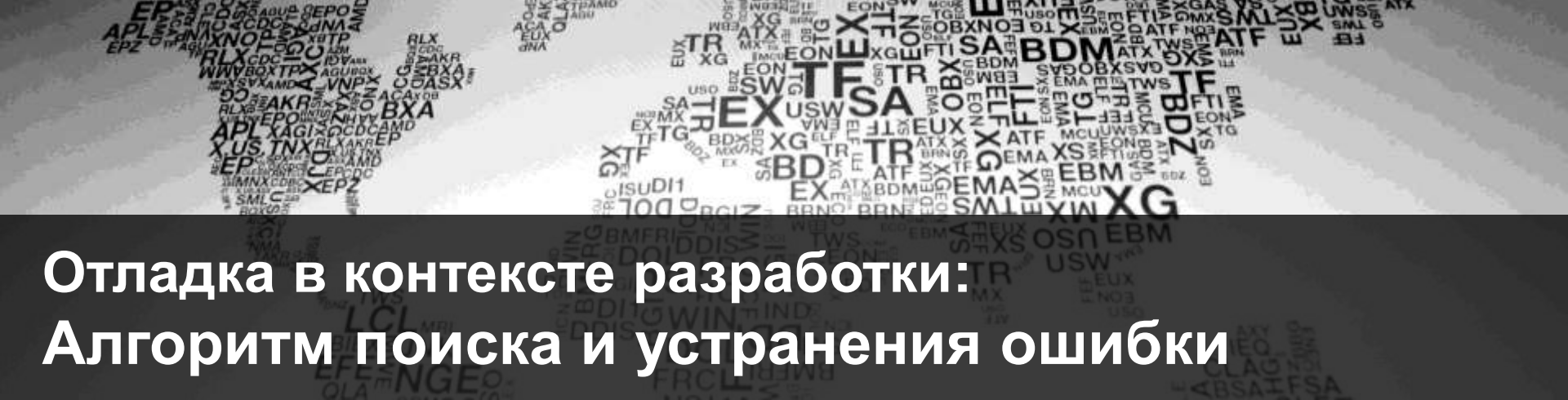




# Отладка в контексте разработки: Алгоритм поиска и устранения ошибки

## Основные шаги. Исследование.

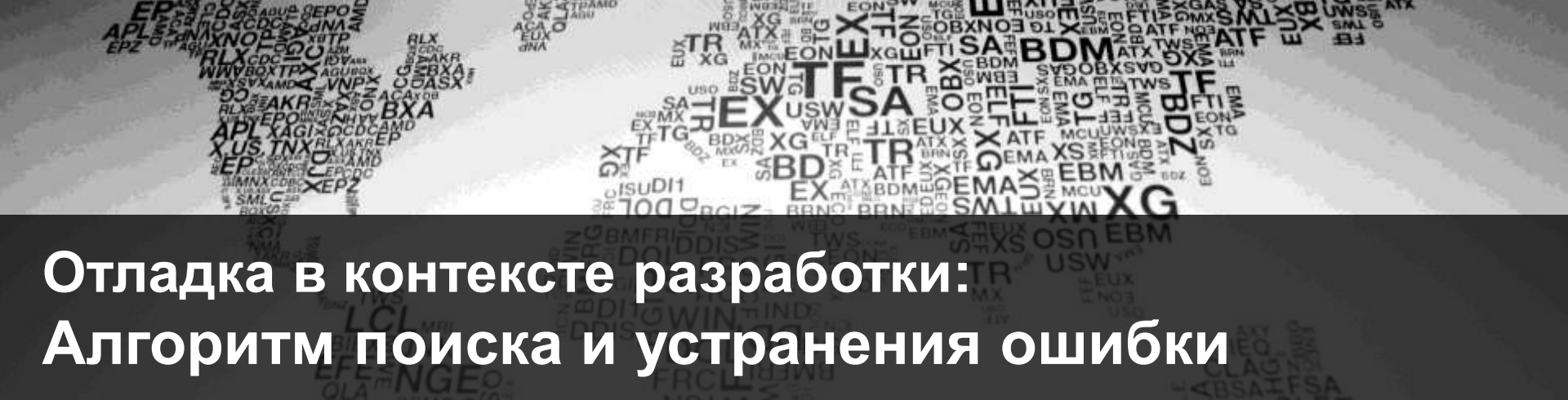
- Длительность этого этапа зависит от успешности предыдущего
- Что помогает исследованию:
  - Код
  - Отладчик и другие инструменты отладки
  - Анализ требований и документации к продукту
  - Описания коммитов
  - Общение с коллегами
- Формальный результат: ясное понимание причины ошибки или публикация описания ошибки в *Системе отслеживания ошибок*



# Отладка в контексте разработки: Алгоритм поиска и устранения ошибки

## Основные шаги. Исправление.

- Проблему в коде исправить проще, чем проблему в дизайне
- Что помогает исправлению:
  - Отладчик и другие инструменты отладки
  - Общение с коллегами
  - Ясное понимание того, какое поведение является верным
  - Общение с бизнес-экспертами
  - Наличие юнит-тестов
  - Принцип «не навреди» (особенно в преддверии релиза)
- Формальный результат: коммит в *Системе контроля версий*



# Отладка в контексте разработки: Алгоритм поиска и устранения ошибки

## Основные шаги. Проверка.

- Проверка кода начинается до тестирования
- Что помогает проверке:
  - Ревью кода
  - Наличие плана для ручного тестирования
  - Наличие автоматических тестов
  - Возможность дать пощупать фикс опытным коллегам
- Формальный результат: баг закрыт в *Системе отслеживания ошибок*



# Отладка в контексте разработки: Тестирование

Виды тестирования:

- Ручное
- Автоматическое

Этапы тестирования:

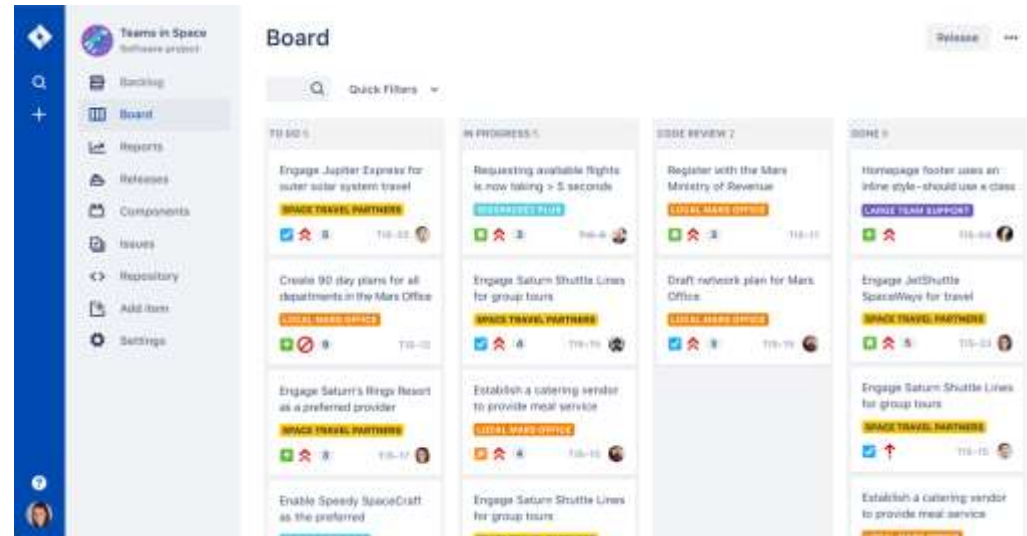
- Unit Testing – тестирование поведения модулей и функций в коде.
- Integration Testing – тестирование взаимодействия нескольких систем программы.
- System Testing – тестирование программы в целом на соответствие требованиям.
- Regression Testing – проверка того, что программа продолжает работать в соответствии с требованиями после сделанных изменений в коде.



# Отладка в контексте разработки: Информационное и программное обеспечение

## Системы отслеживания ошибок

- Распределение задач между разработчиками
- Расстановка приоритетов
- Хранение информации о всех известных ошибках







# Отладка в контексте разработки: Информационное и программное обеспечение

## Системы отслеживания ошибок

Возможности:

- Отслеживание состояния задач по исправлению ошибок
- Хранение базы знаний по всем известным ошибкам

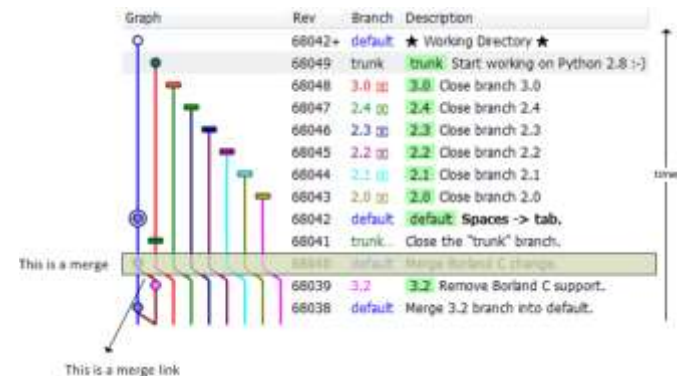
Состояние абстрактной проблемы:

- Проблема неизвестна (нет в *Системе отслеживания*)
- Проблема известна и находится в работе у кого-то
- Проблема известна и ждет, пока до нее дойдут руки
- Проблема известна, но не планируется к исправлению
- Проблема известна, и уже исправлена

# Отладка в контексте разработки: Информационное и программное обеспечение

## Системы контроля версий

- Хранение полной истории изменений кода
- Анализ изменений в коде
- Управление версиями программы





# Отладка в контексте разработки: Информационное и программное обеспечение

## Системы контроля версий

Возможности:

- Поиск конкретной версии, если известны шаги для воспроизведения проблемы (git bisect)
- Поиск изменения (коммита), если известна версия, в которой появилась проблема
- Поиск изменений в коде, если известен файл, в котором проблема
- Описания коммитов могут раскрыть причины, почему подозрительный код написан так, а не иначе



# Отладка в контексте разработки: Информационное и программное обеспечение

## Логирование

Логи – файлы, содержащие диагностическую информацию:

- Действия пользователя перед появлением проблемы
- Изменения внутреннего состояния программы, предшествующие проблеме
- Переменные среды окружения
- Приложения, выполняющиеся на компьютере пользователя
- Информация о компьютере пользователя

(модель процессора, размер монитора, объем памяти и т.д.)





# Отладка в контексте разработки: Информационное и программное обеспечение

## Логирование

Возможности хорошего логгера:

- Несколько уровней логирования
- Ротация лог-файлов
- Возможность записи сообщений не только в файлы
- Поток-безопасность
- Асинхронное логирование
- Настройка формата записей





# Отладка в контексте разработки: Информационное и программное обеспечение

## Уровни логирования

Debug	Подробная внутренняя информация о работе программы.
Information	Краткая информация об изменении состояния программы.
Warning	Программа находится в неожиданном или нестандартном состоянии. Лучше не игнорировать.
Error	Явная ошибка в работе программы. Программа в целом продолжает работу.
Fatal	Ошибка, приводящая к неработоспособности всей программы или подсистемы.

# Отладка в контексте разработки: Информационное и программное обеспечение

## Логирование

Пример лог-файла:

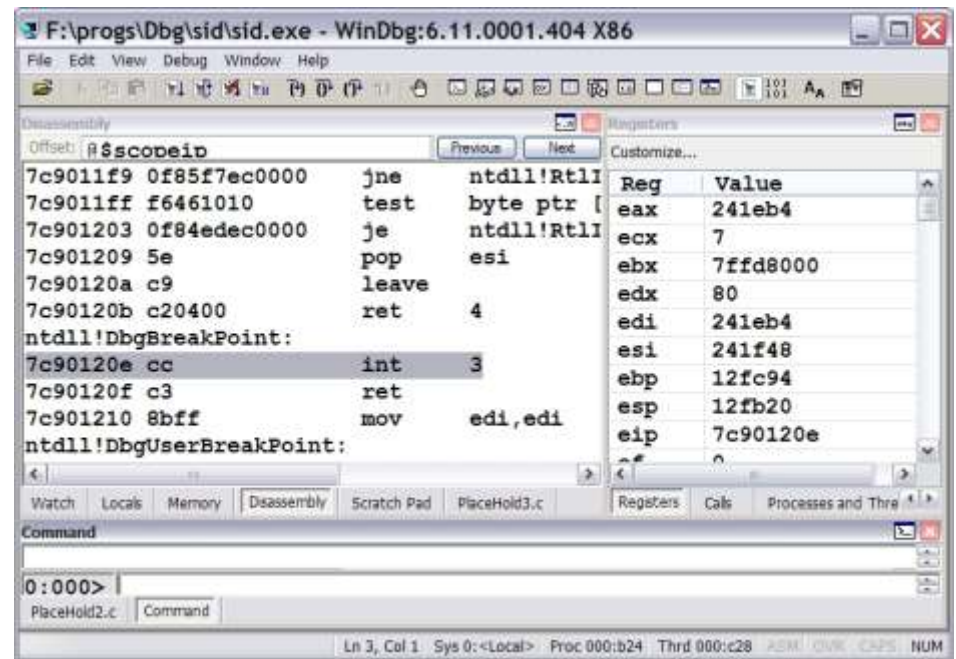
```
app_2022_09_09_14_47_00.log (C:\Users\adenis) - GVIM1
1 2022-09-09 14:47:16.345+03:00 [INFO] [0x00006c04] [core] App.EXE/JETS: creating context for session 75
2 2022-09-09 14:47:29.071+03:00 [WARN] [0x000075b4] [net] 172.26.130.45: Destination host unreachable.$
3 2022-09-09 14:47:30.309+03:00 [TRACE] [0x000023ab] [net] 35 bytes sent to 172.26.130.111$
4 2022-09-09 14:47:30.317+03:00 [INFO] [0x000075b4] [util] Image conversion from Image.JPG to Image.PNG succeeded$
```



# Инструменты отладки ПО: Отладчик (Debugger)

Возможности:

- Приостановка программы
- Пошаговое исполнение
- Отслеживание и изменение значений переменных и произвольных участков памяти
- Настройка точек останова



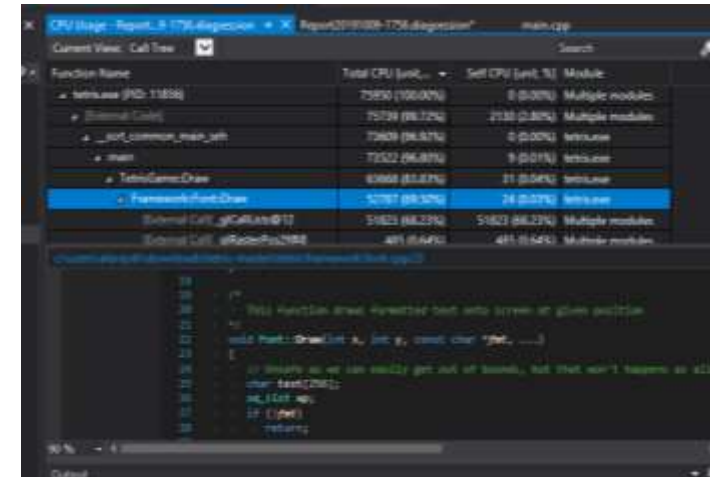


# Инструменты отладки ПО: Профилировщик (Profiler)

Возможности:

- Анализ отдельных функций и строк кода.
- Анализ конкретного временного диапазона.
- Анализ конкретного потока программы.
- Определяет количество вызовов и время, которое проводится внутри функции.

Только профилировщик способен подсказать, где скрываются узкие места вашей программы.



Примеры:

- Visual Studio
- Windows Performance Analyzer



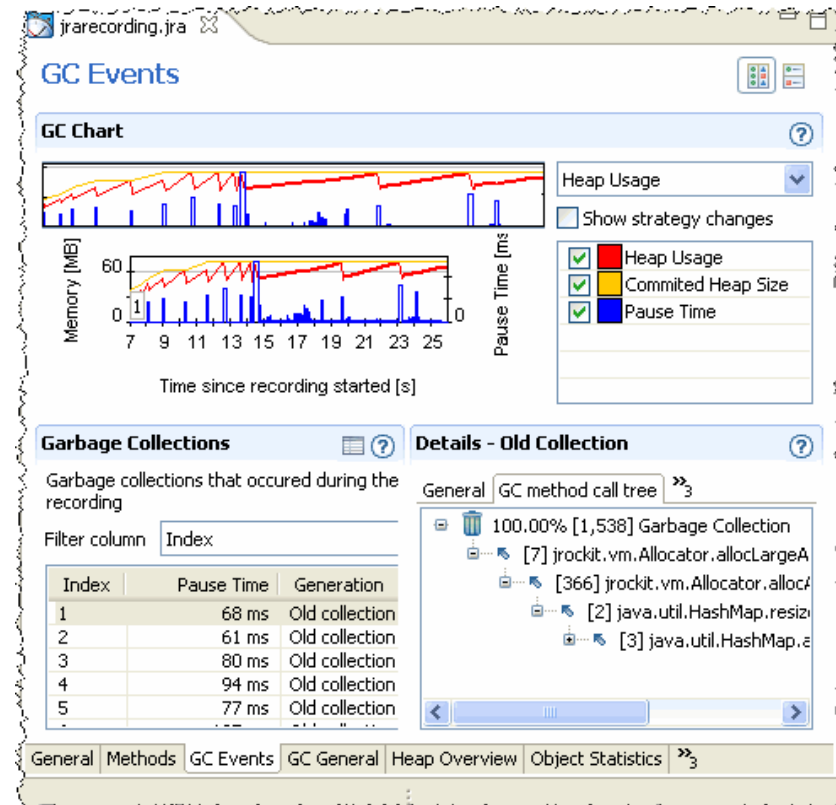
# Инструменты отладки ПО: Детектор утечек памяти

Возможности:

- Выявление “утечки” памяти (в языках с ручным управлением памятью)
- Выявление неоптимального использования памяти

Примеры:

- Visual Studio
- Visual Leak Detector
- Intel Inspector



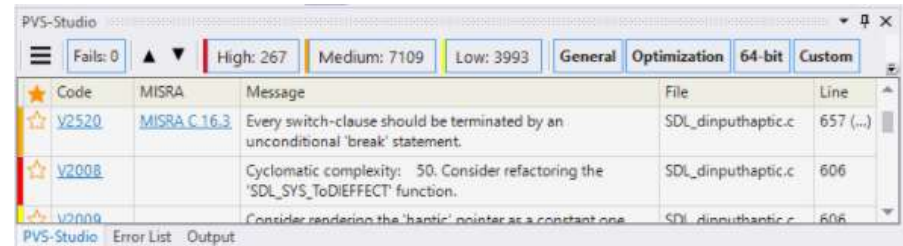
# Инструменты отладки ПО: Статический анализатор кода

Статический анализатор:

- Анализирует исходный код на предмет логических ошибок и подозрительных конструкций.

Примеры:

- PVS-Studio
- Visual Studio Static Analysis



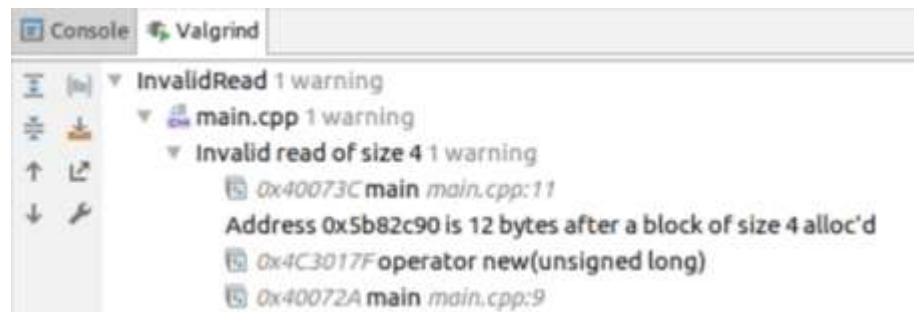
# Инструменты отладки ПО: Динамический анализатор кода

Динамический анализатор:

- Анализирует работающую программу на предмет корректной работы с памятью и с ресурсами системы.

Примеры:

- Application Verifier
- Address Sanitizer



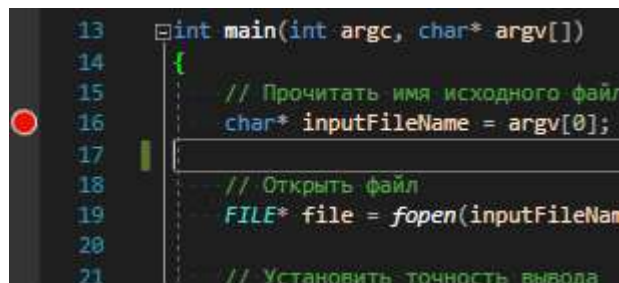
# Отладчик Visual Studio

- Точки остановки (Breakpoints)
- Выполнение программы
- Стек вызовов (Call Stack)
- Просмотр переменных (Watch window)
- Debug и Release конфигурации



# Отладчик Visual Studio: Точки останова: типы

Simple Breakpoint – остановка программы при выполнении конкретной строки кода.



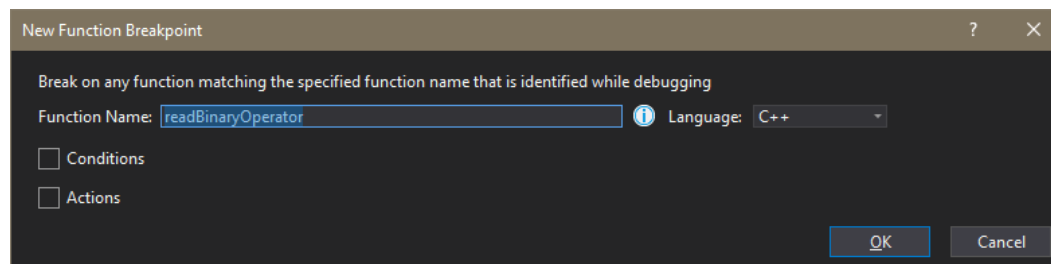
```
13  int main(int argc, char* argv[])
14  {
15      // Прочитать имя исходного файла
16      char* inputFileName = argv[0];
17
18      // Открыть файл
19      FILE* file = fopen(inputFileName, "r");
20
21      // Установить точность вывода
```

The image shows a screenshot of the Visual Studio code editor. A red circle, representing a simple breakpoint, is set on line 16 of the code. The code is written in C++ and includes comments in Russian. The line numbers 13 through 21 are visible on the left side of the editor.



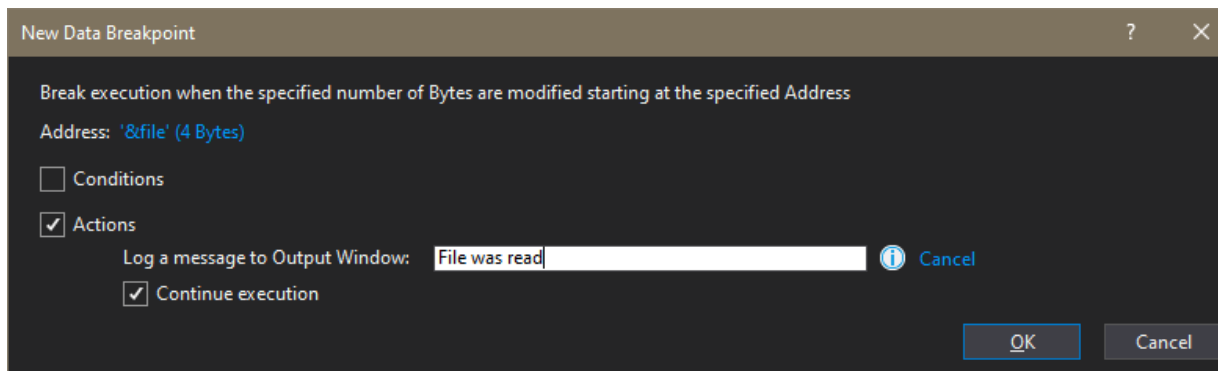
# Отладчик Visual Studio: Точки останова: типы

Function Breakpoint – остановка программы при вызове функции с указанным именем. Удобно использовать для того, чтобы остановить программу при вызове любой функции класса.



# Отладчик Visual Studio: Точки останова: типы

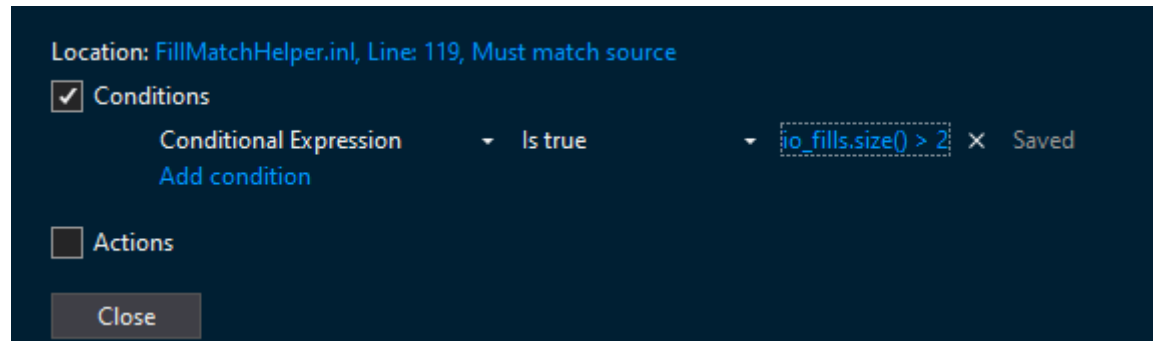
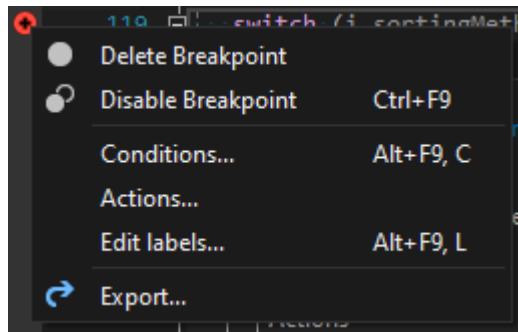
Data Breakpoint – остановка программы при изменении памяти по указанному адресу. Обычно ограничено размером указателя (4 или 8 байт).



# Отладчик Visual Studio: Точки останова: настройка

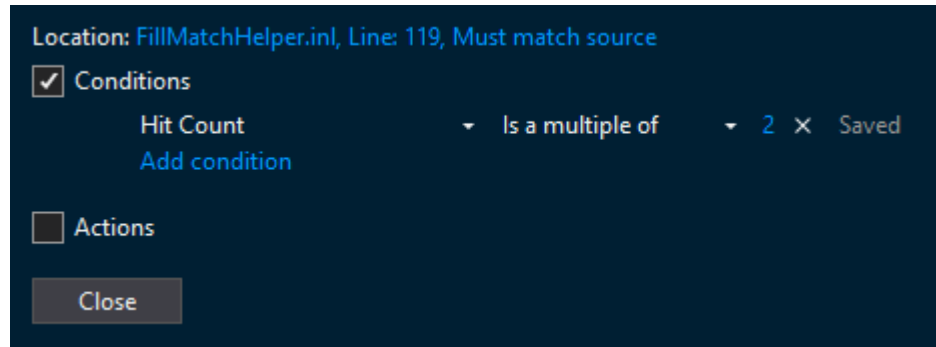
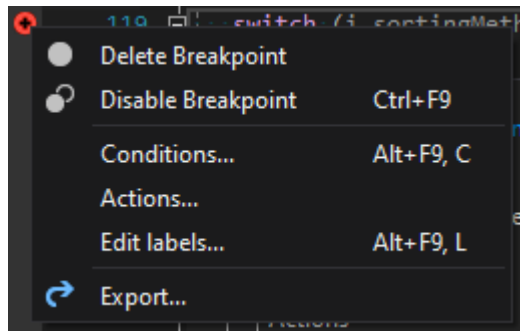
Conditions: Conditional Expression – остановка программы только при выполнении указанного условия.

Синтаксис условий – упрощенный C++.



# Отладчик Visual Studio: Точки останова: настройка

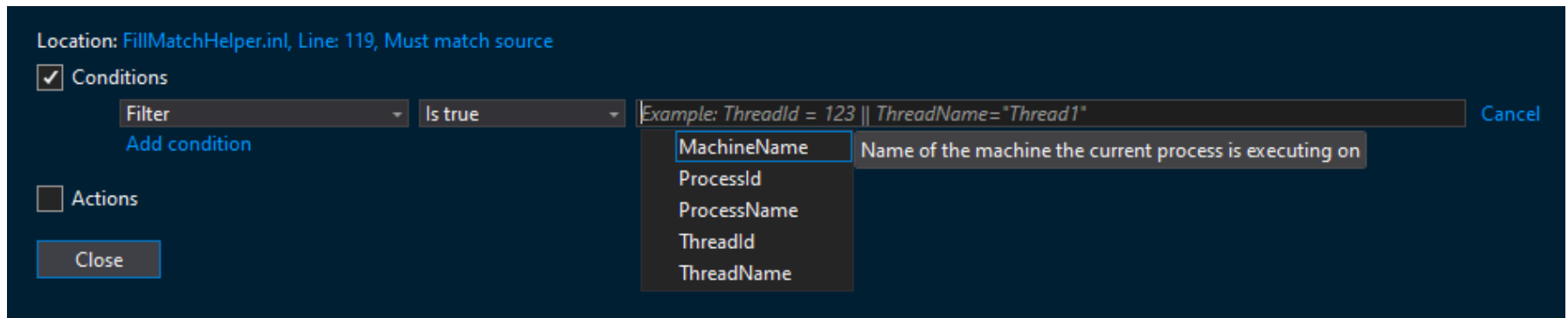
Conditions: Hit Count – остановка программы только при определенном количестве попаданий в точку останова.





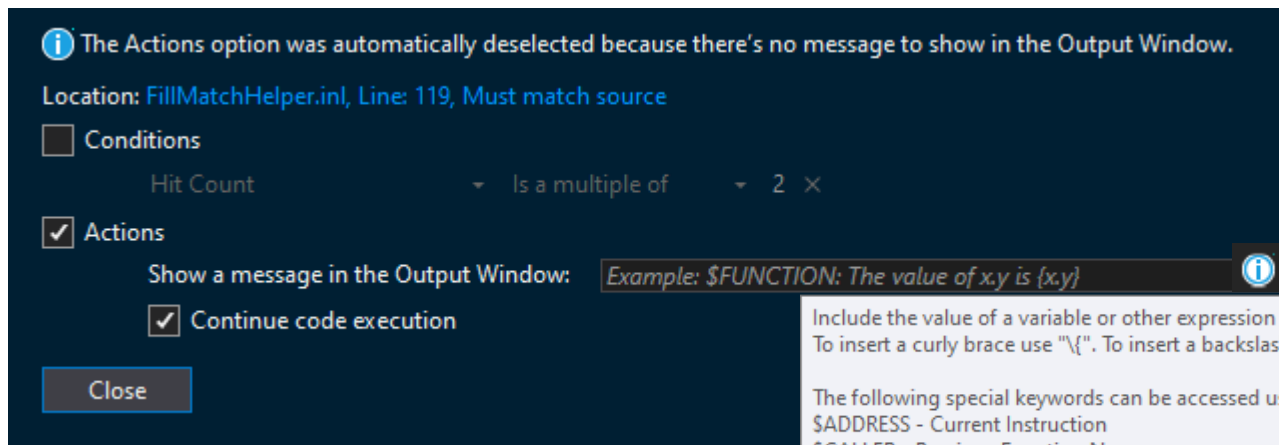
# Отладчик Visual Studio: Точки останова: настройка

Conditions: Filter – остановка программы только при выполнении дополнительных критериев (например, только в конкретном потоке).



# Отладчик Visual Studio: Точки останова: настройка

Actions – напечатать сообщение в окно Output и, опционально, продолжить работу.



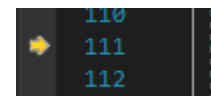
Include the value of a variable or other expression by placing it in curly braces (Example: "The value of x is {x}"). To insert a curly brace use "{". To insert a backslash, use "\\".

The following special keywords can be accessed using "\$"

- \$ADDRESS - Current Instruction
- \$CALLER - Previous Function Name
- \$CALLSTACK - Call Stack
- \$FILEPOS - The current file and line position
- \$FUNCTION - Current Function Name
- \$PID - Process Id
- \$PNAME - Process Name
- \$TICK - The number of milliseconds that have elapsed since the system was started, up to 49.7 days
- \$TID - Thread Id
- \$TNAME - Thread Name

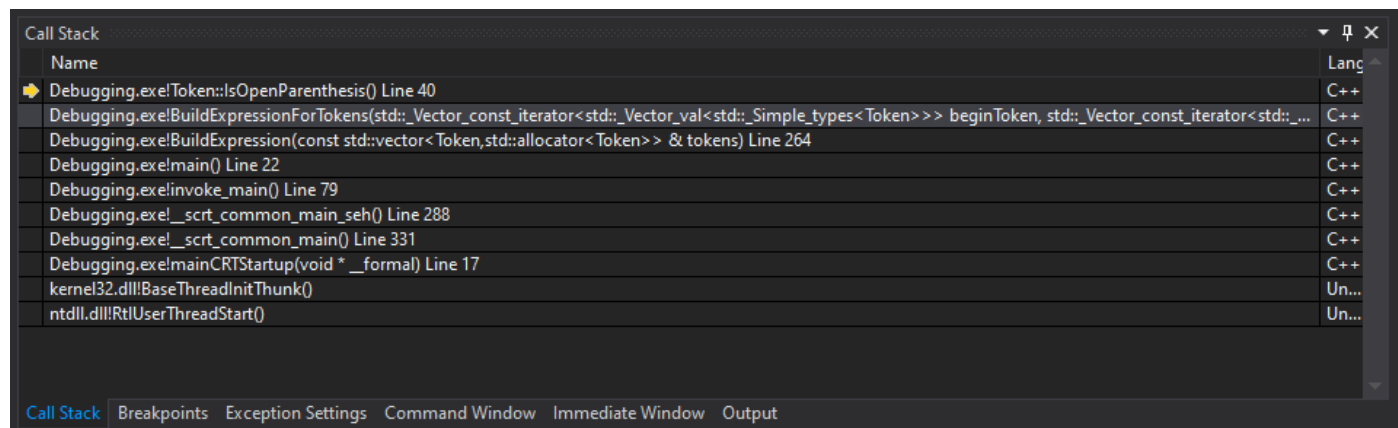
# Отладчик Visual Studio: Выполнение программы

- Continue – продолжить работу программы до следующей остановки
- Break All – остановить все потоки программы прямо сейчас
- Step Into – выполнить следующую операцию, ничего не пропуская
- Step Over – выполнить следующую строку программы целиком
- Step Out – выполнить текущую функцию до конца
- Run to Cursor – выполнить программу до выбранной строки
- Set Next Statement – нарушить поток исполнения и немедленно перейти в выбранную точку программы



# Отладчик Visual Studio: Стек вызовов

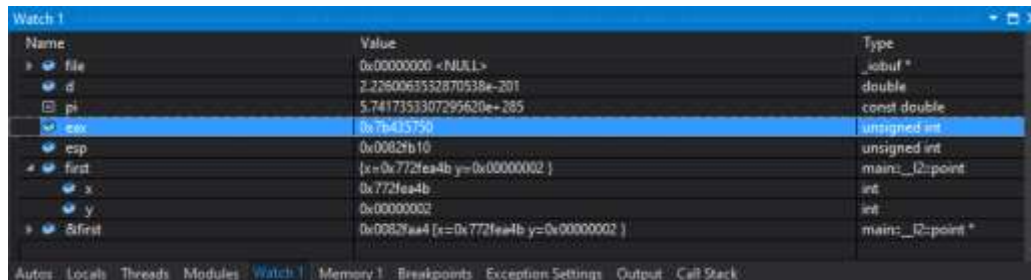
- Показывает полную цепочку вызовов функций, которая привела к текущему состоянию программы
- Сверху – текущая точка исполнения программы
- В середине – точка входа main()



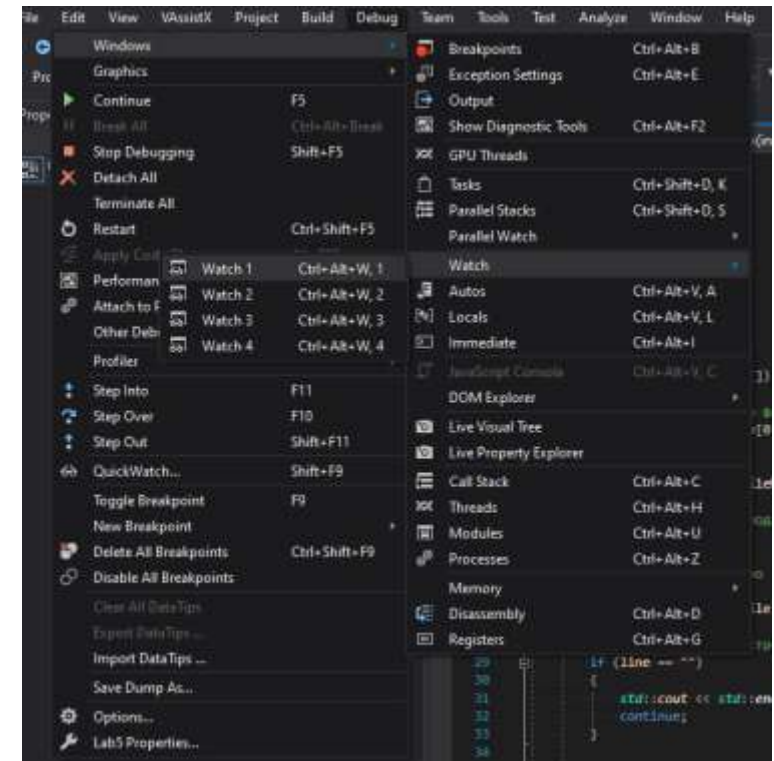


# Отладчик Visual Studio: Просмотр переменных

- Просмотр значений глобальных и локальных переменных
- Просмотр значений простых выражений
- Изменение значений переменных
- Форматирование значений в удобной форме

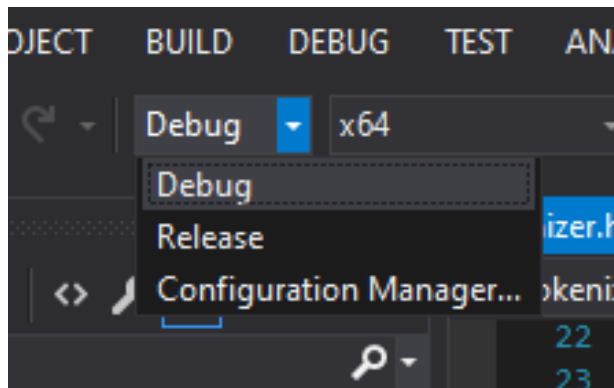


Name	Value	Type
file	0x00000000 <NULL>	_ioobuf *
d	2.2260063532870538e-201	double
pi	5.7417353307295620e+285	const double
eax	0x76435750	unsigned int
esp	0x0082fb10	unsigned int
first	{x=0x772fea4b y=0x00000002 }	main::_I2::point
x	0x772fea4b	int
y	0x00000002	int
Sfirst	{x=0x0082faa4 {x=0x772fea4b y=0x00000002 }	main::_I2::point *



# Отладчик Visual Studio: Debug и Release

- В Debug конфигурации код не оптимизируется и содержит избыточную информацию для максимального удобства отладки.
- В Release конфигурации код оптимизирован, а отладка при этом может быть затруднена или вовсе невозможна.





# Дампы памяти (memory dumps)

- Назначение
- Создание
- Анализ (с PDB и без)

# Дампы памяти (memory dumps): Назначение

Содержат информацию о состоянии программы в определённый момент времени:

- Полный/частичный снимок памяти
- Поток приложения
  - стек вызовов
  - значения регистров процессора
  - значения локальных переменных
- Информация об ошибке или исключении





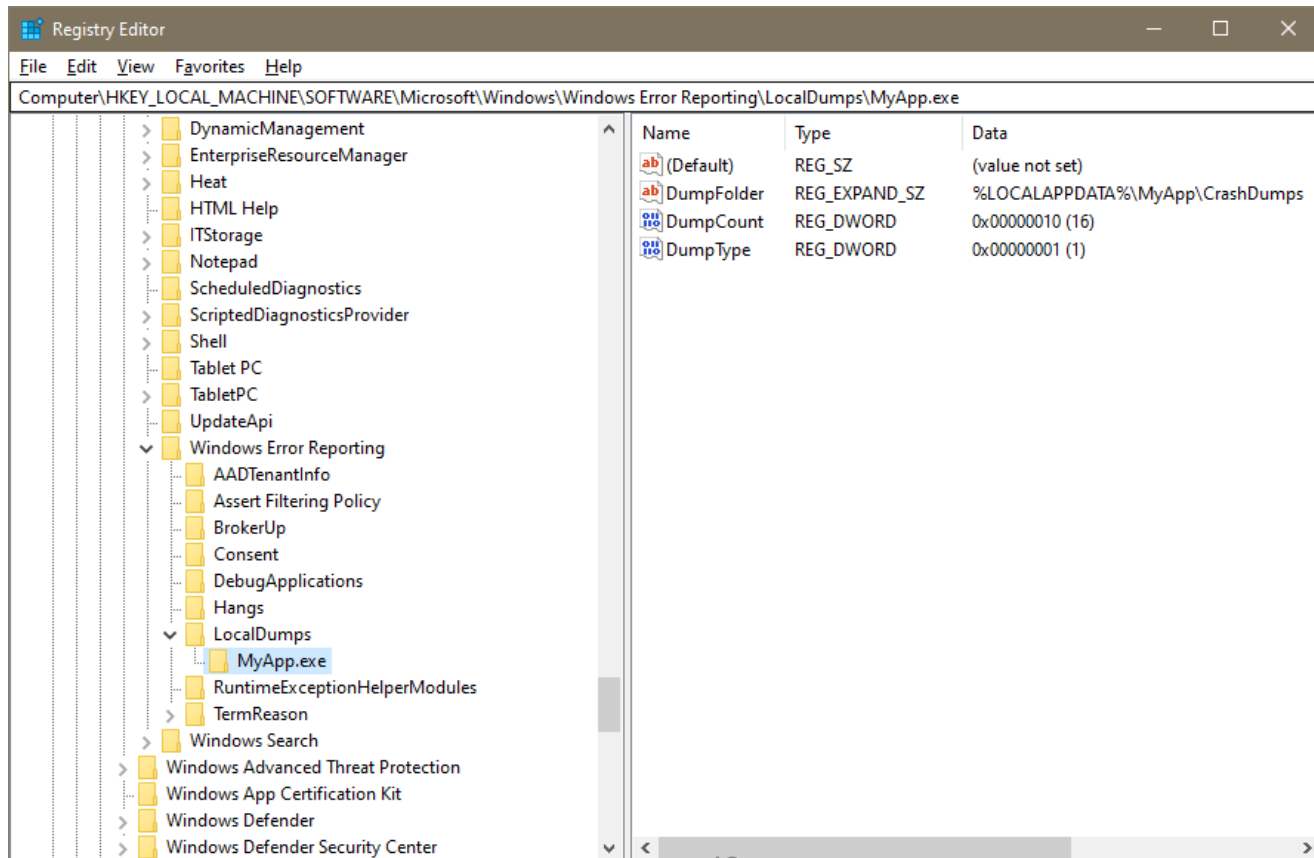
# Дампы памяти (memory dumps): Создание (вариант 1)

```
#include <dbghelp.h>
```

```
BOOL WINAPI MiniDumpWriteDump( HANDLE hProcess,  
    DWORD ProcessId,  
    HANDLE hFile,  
    MINIDUMP_TYPE DumpType,  
    PMINIDUMP_EXCEPTION_INFORMATION ExceptionParam,  
    PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam,  
    PMINIDUMP_CALLBACK_INFORMATION CallbackParam );
```

```
#include <windows.h>  
LONG TopLevelFilter(_EXCEPTION_POINTERS *pExceptionInfo )  
{  
    return GenerateDump(pExceptionInfo);  
}  
void SetExceptionHook()  
{  
    ::SetUnhandledExceptionFilter(TopLevelFilter );  
}
```

# Дампы памяти (memory dumps): Создание (вариант 2)

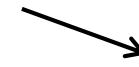


# Дампы памяти (memory dumps): Анализ

WinDbg



v1.0-20120505-122822-5836-7776.dmp  
v1.0-20120505-122943-5600-4628.dmp  
v1.0-20120505-123017-2680-6400.dmp



Visual Studio

```
Microsoft (R) Windows Debugger Version 6.12.0002.633 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\Users\romangol\AppData\Local\Temp\AppName\v1.0-20120505-123017-2680-6400.dmp]
User Mini Dump File: Only registers, stack and portions of memory are available

Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.
* Use symfix to have the debugger choose a symbol path.
* After setting your symbol path, use .reload to refresh symbol locations.
*****
Executable search path is:
Windows 7 Version 7601 (Service Pack 1) MP (4 procs) Free x86 compatible
Product: WinNt, suite: SingleUserTS
Machine Name:
Debug session time: Sat May 5 12:30:50.000 2012 (UTC + 4:00)
System Uptime: not available
Process Uptime: 0 days 0:00:53.000

This dump file has an exception of interest stored in it.
The stored exception information can be accessed via _EXCEPTION_RECORD
(77e19b01). Integer divide-by-zero - code c0000094 (first second chance not available)
eax=00000000 ebx=00380d78 ecx=00000000 edx=00000000 esi=00380d38 edi=0017ebd8
esp=77e40c22 ebp=0017e898 ibp=0017e8a8 iopl=0         nv up ei pl zr na pe nc
cs=0023  eip=002b  da=002b  wa=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found. Defaulted to export symbols for ntdll.dll -
ntdll!ZwGetContextThread+0x12:
77e40c22.83c404      add     esp,4
```

Minidump File Summary  
10/11/2019 12:00:28 PM

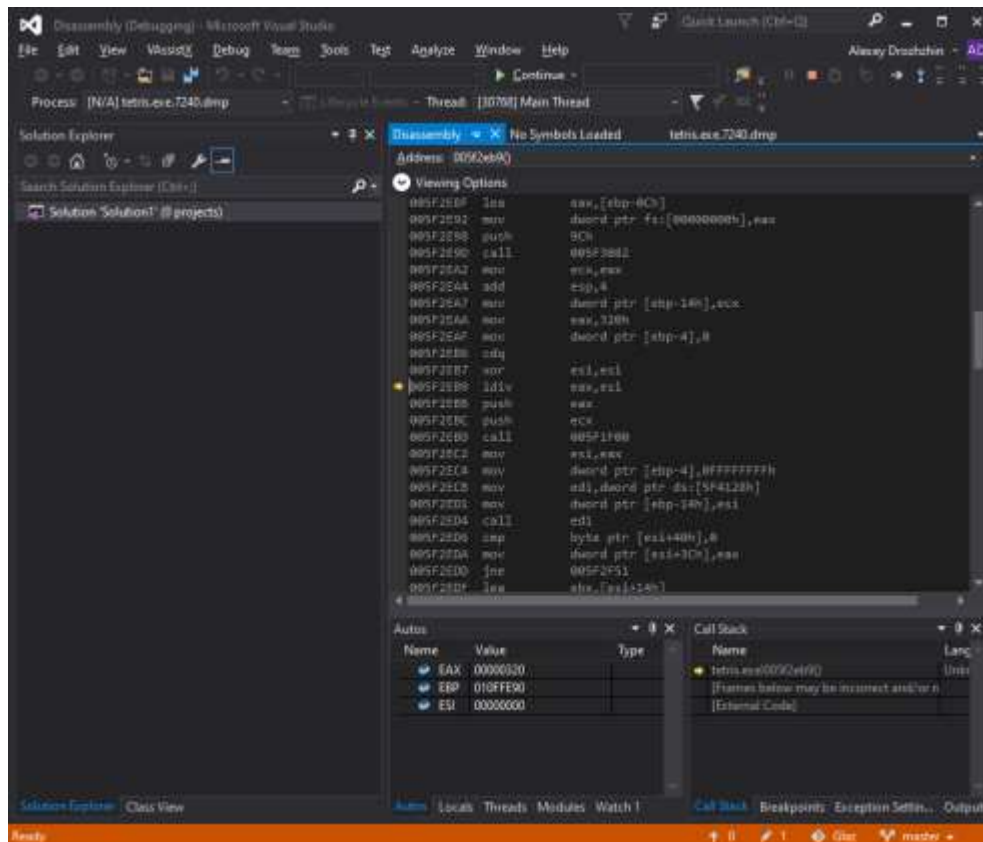
Category	Value
Dump File	ntdll.exe (77e19b01) (C:\Users\romangol\AppData\Local\Temp\AppName\v1.0-20120505-123017-2680-6400.dmp)
Last Write Time	10/11/2019 12:30:50 PM
Process Name	ntdll.exe
Process Architecture	x86
Exception Code	0xc0000094
Exception Information	The thread tried to divide an integer value by an integer division of zero.
Heap Information	Not Present
Error Information	Not Present

Module Name	Module Version	Module Path
ntdll.exe	6.0.6002.18005	C:\Users\romangol\AppData\Local\Temp\AppName\v1.0-20120505-123017-2680-6400.dmp
kernel32.dll	6.0.6002.18005	C:\Windows\System32\kernel32.dll

Output  
Show output from: General  
We were unable to automatically populate your Visual Studio Team Services accounts.  
The following error was encountered: VSERR03: You are not authorized to access https://msdn.microsoft.com/...

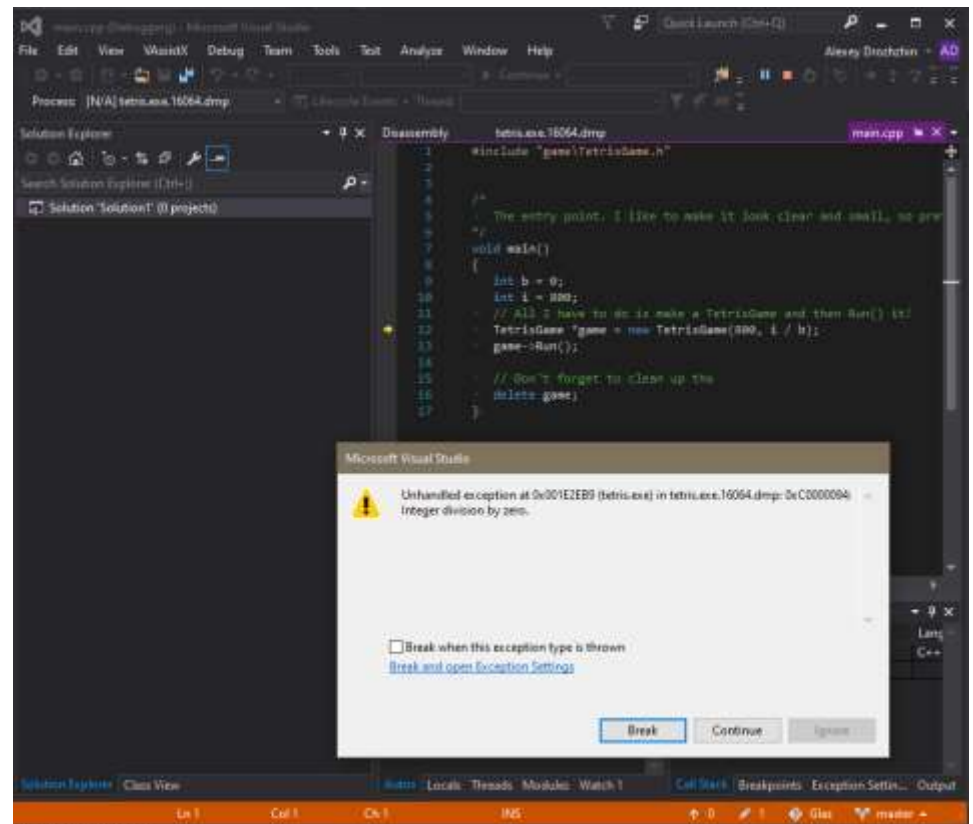
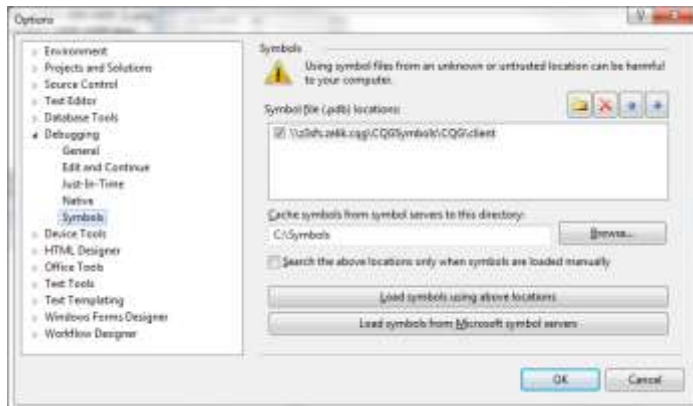
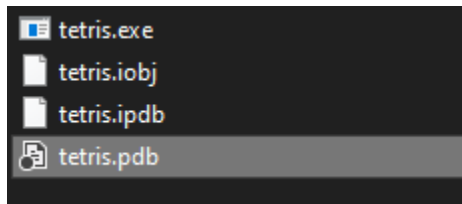
Ln 0, Col 0 - Sys 0: C:\User - Proc 000a78 - Thrd 0001900 - ASM - QWE - CAPS - NUM

# Дампы памяти (memory dumps): Анализ (без PDB)





# Дампы памяти (memory dumps): Анализ (с PDB)





# Типовые ошибки

- Общие закономерности
- Типы ошибок
  - Access violation
  - Memory leak
  - Heap corruption
  - Stack overflow

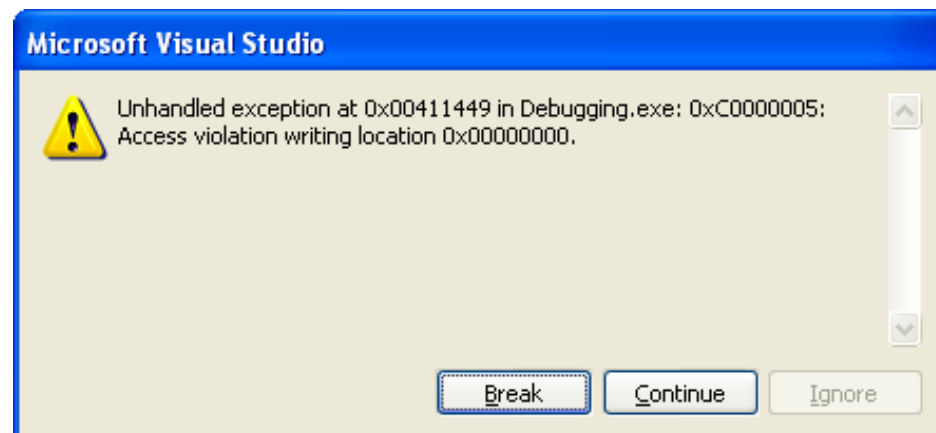


# Типовые ошибки: Общие закономерности

- Ошибку исправить раньше легче чем позже.
- Ошибка чаще всего в вашем коде.
- Изредка, ошибка бывает в сторонней библиотеке или фреймворке.
- Чем популярнее сторонняя библиотека, там меньше в ней ошибок.
- Ошибки *случаются* в компиляторе, в ОС и в драйверах.

# Типовые ошибки

## Access violation



### Причины:

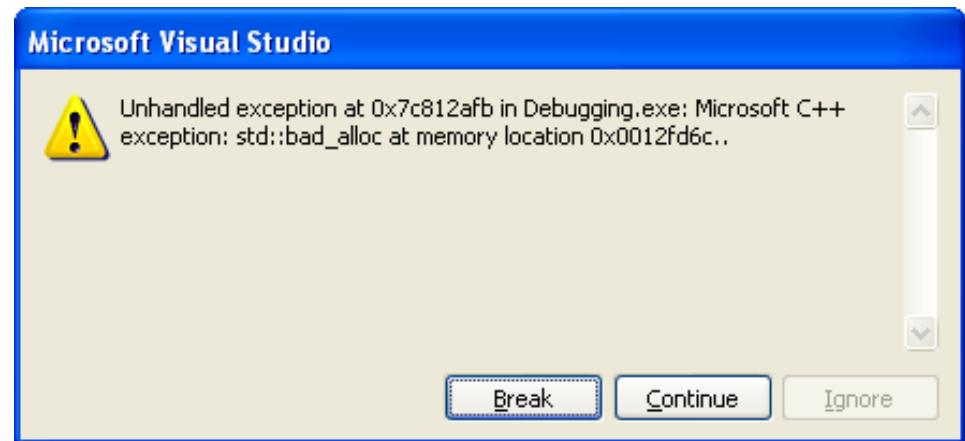
- Неинициализированные переменные и члены класса
- Висячие указатели

# Типовые ошибки

## Memory leak

### Причины:

- Забытый delete (без умных указателей)
- Циклические ссылки (с умными указателями)
- Невиртуальный деструктор в базовом классе



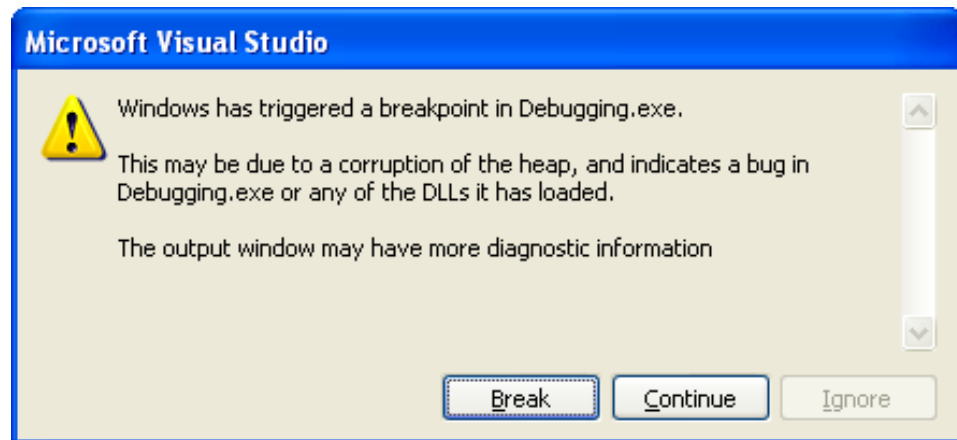


# Типовые ошибки

## Heap corruption

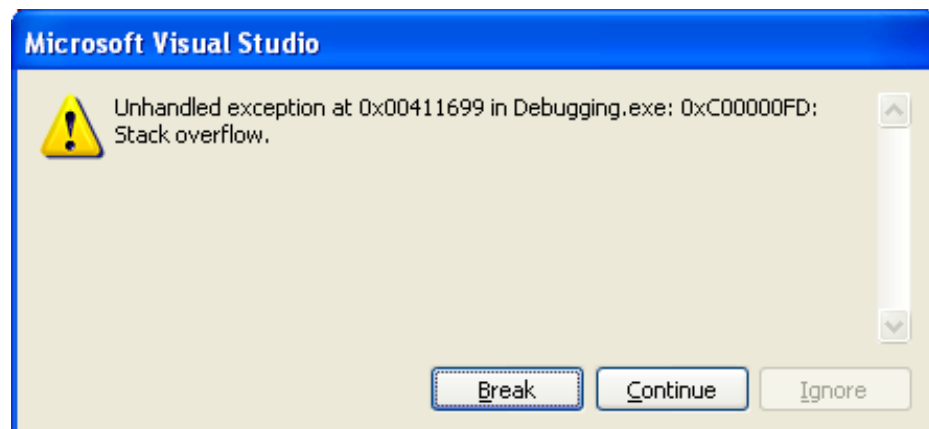
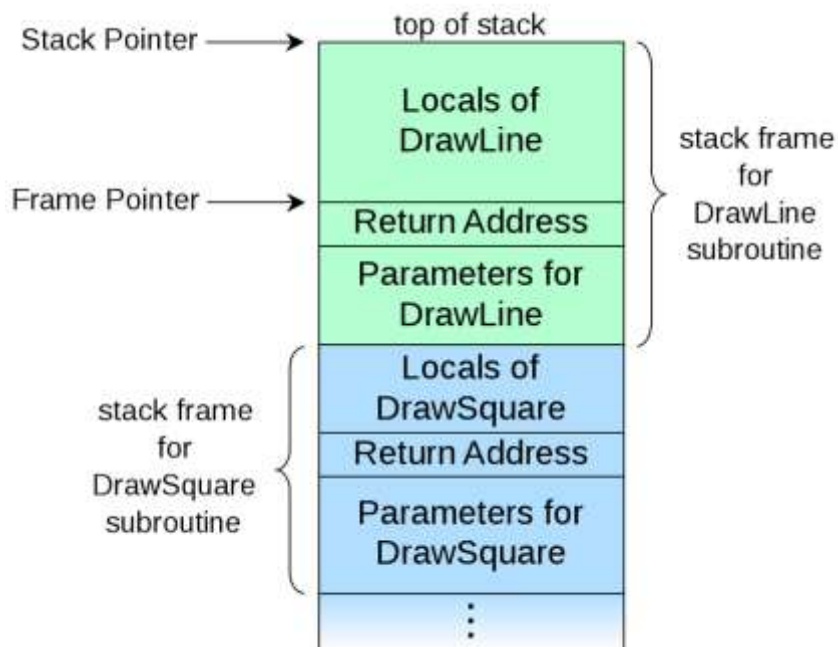
### Причины:

- Некорректная работа с памятью
- Двойное удаление объекта
- Переполнение массива



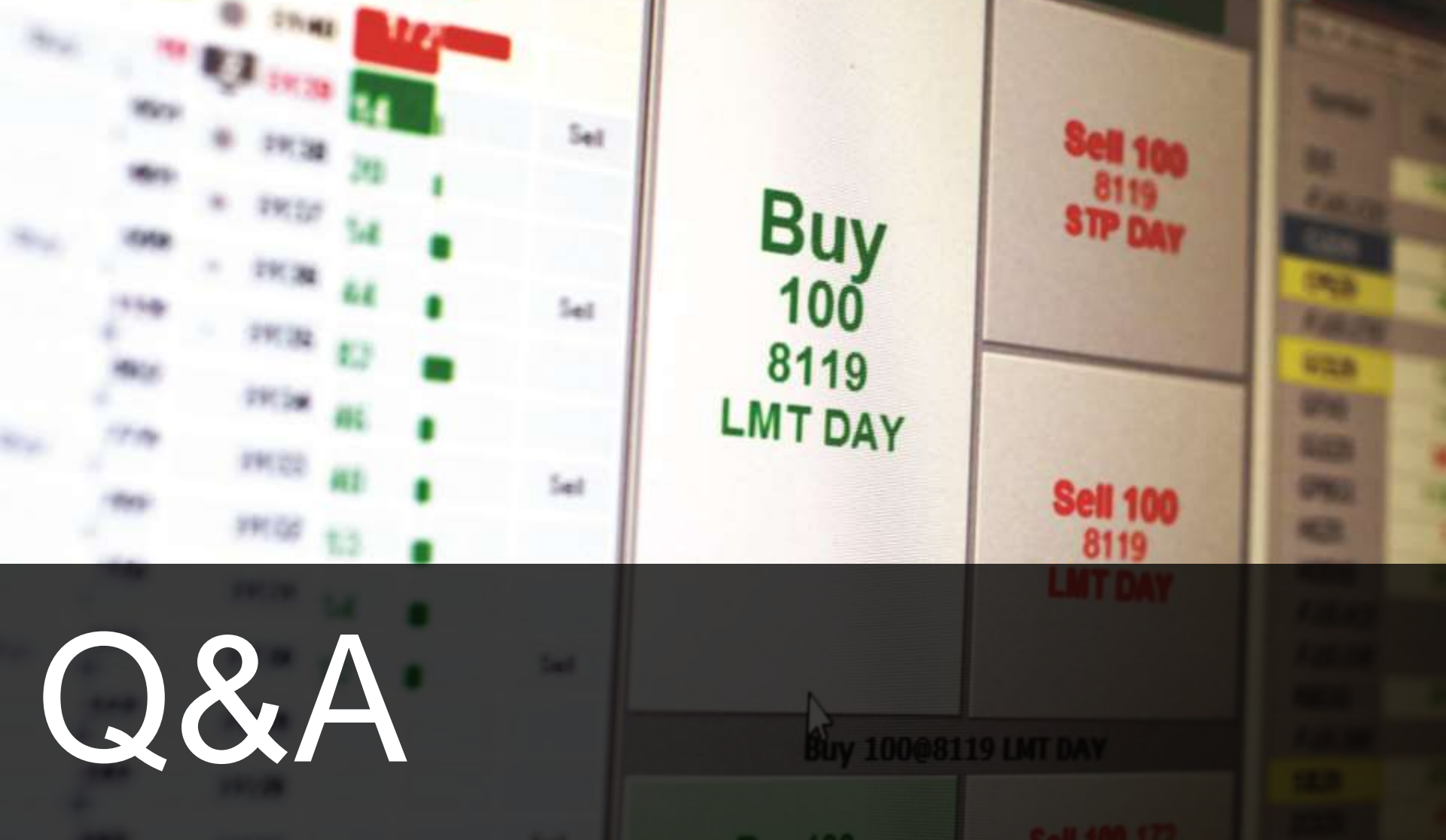
# Типовые ошибки

## Stack overflow



### Причины:

- Бесконечная рекурсия
- Глубокая рекурсия
- Создание больших массивов на стеке



# Q&A

# Очень интересная литература



- Tarik Soulati, Inside Windows Debugging: A Practical Guide to Debugging and Tracing Strategies in Windows
- Д. Роббинс. Поиск и устранение ошибок в программах под Windows
- С. Макконнелл. Совершенный код
- Д. Востоков, Memory Dump Analysis Anthology