

# Искусство отладки

Евгений Козлов



# Структура лекции

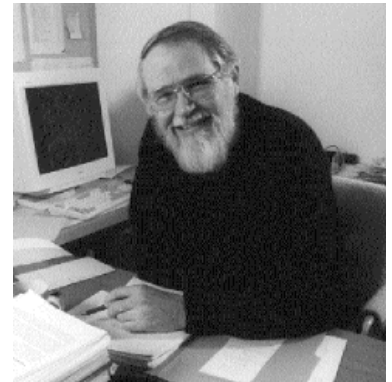
## Основные разделы и подразделы.

- ❑ Введение
- ❑ Подготовка и планирование
  - Неизбежность отладки
  - Поддержка процессом
  - Информационное и программное обеспечение
- ❑ Инструменты отладки ПО
  - Отладчик
  - Профилировщик
  - Детектор утечек памяти
- ❑ Тестирование
- ❑ Отладчик Visual Studio
  - Точки останова
  - Выполнение программы
  - Стек вызовов
  - Просмотр переменных
  - Debug и Release
- ❑ Crash dumps
- ❑ Типовые ошибки
- ❑ Q&A
- ❑ Литература

# Введение

**Отладка** – это поиск причин и устранение ошибок в программе.

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



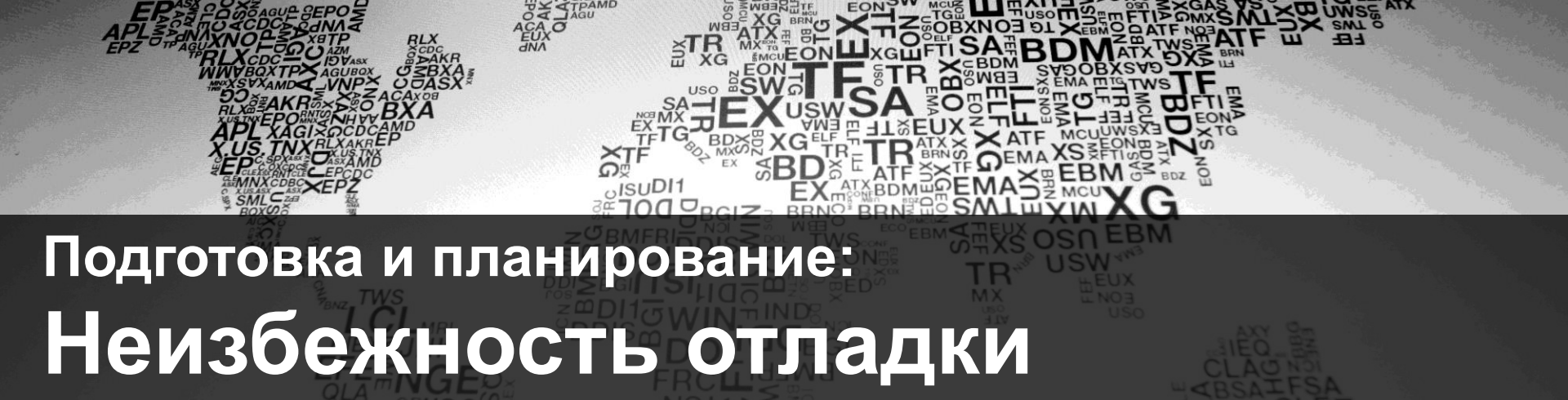
**Брайан Керниган**



# Подготовка и планирование

- Неизбежность отладки
- Поддержка процессом
- Алгоритм поиска и устранения ошибки
- Информационное и программное обеспечение



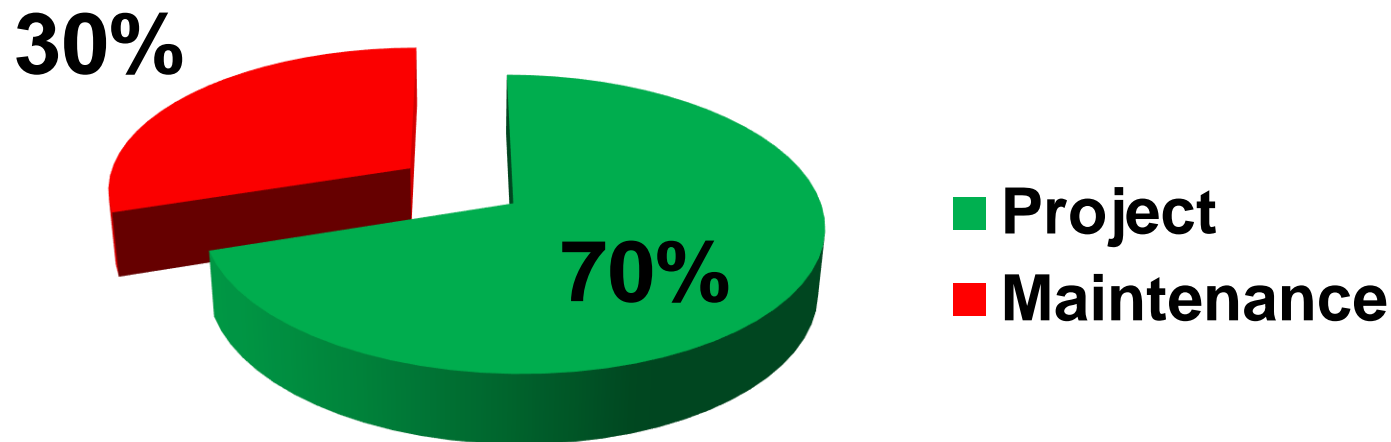


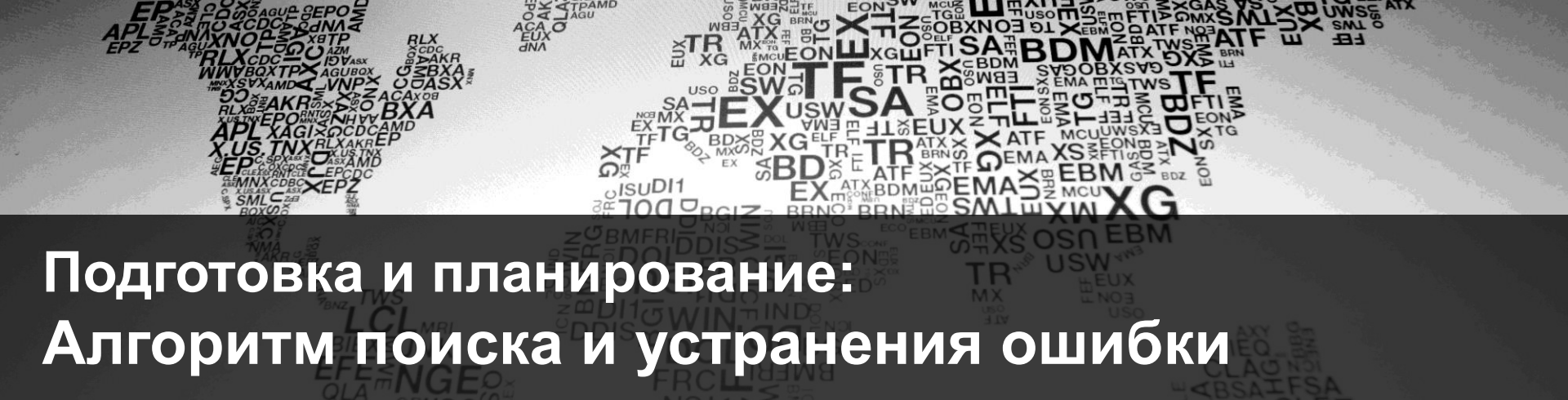
# Подготовка и планирование: Неизбежность отладки

- Человеческий фактор
- Недостаток знаний и навыков
- Сжатые сроки
- Неясные требования

# Подготовка и планирование: Поддержка процессом

До 20-40% ресурсов тратится на поиск и исправление ошибок в существующем ПО





# Подготовка и планирование: Алгоритм поиска и устранения ошибки

## Основные шаги:

Воспроизведение (reproducing)

Поиск ошибки (investigation)

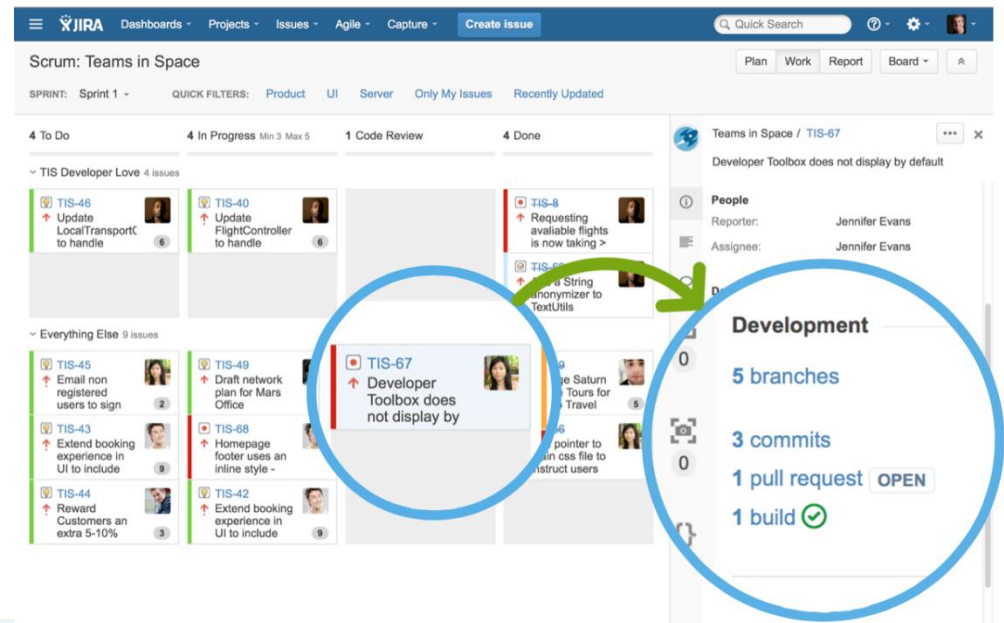
Исправление (fixing)

Проверка решения (testing)

# Подготовка и планирование: Информационное и программное обеспечение

## Системы отслеживания ошибок

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





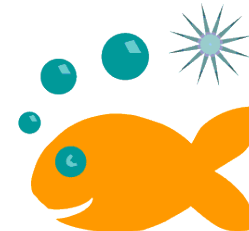
# Подготовка и планирование: Информационное и программное обеспечение

## Системы контроля версий

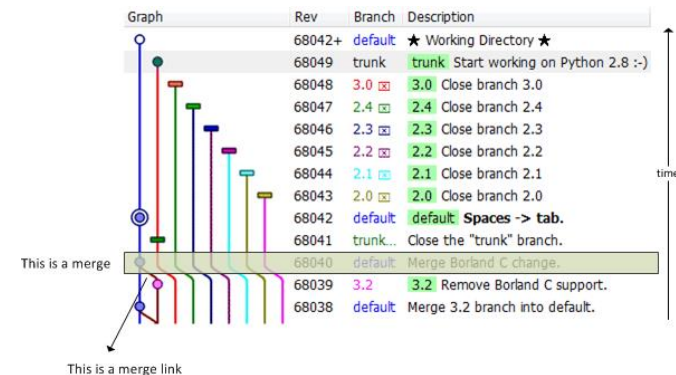
- Хранение полной истории изменений кода
- **Анализ изменений в коде**
- Управление версиями программы



mercurial

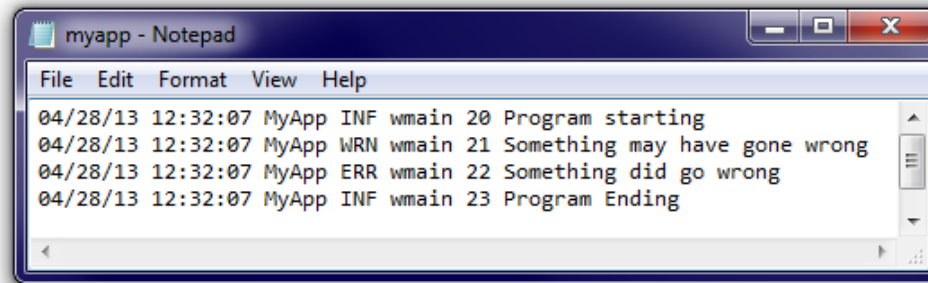


CVS



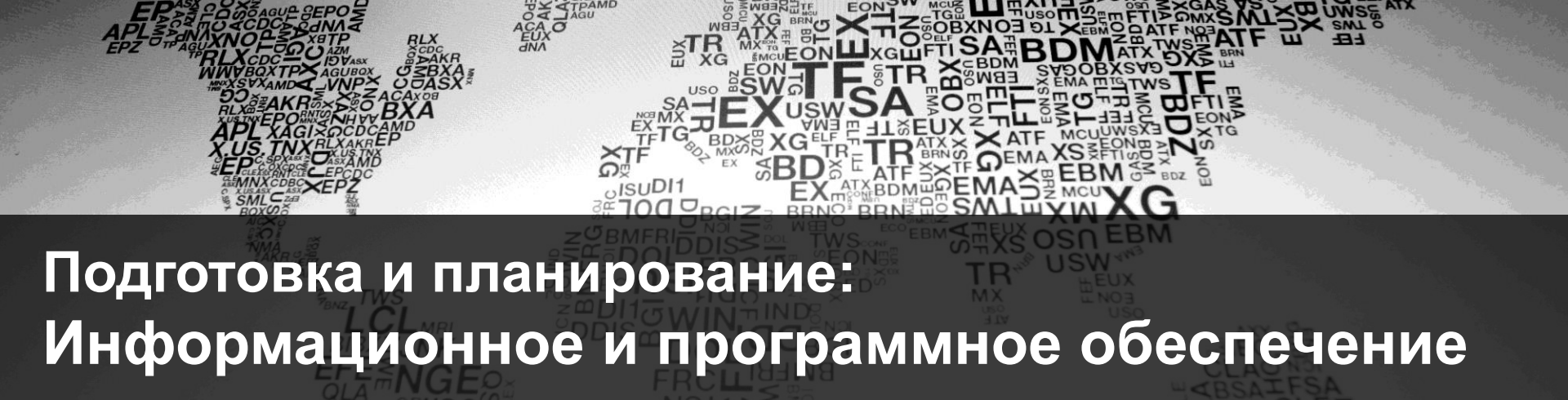
# Подготовка и планирование: Информационное и программное обеспечение

## Логирование



### Возможности хорошего логгера:

- Уровни логирования и фильтрация сообщений
- Ротация лог-файлов
- Возможность записи сообщений не только в файлы
- Потокбезопасность
- Асинхронное логирование
- Гибкое форматирование и конфигурация логов



# Подготовка и планирование: Информационное и программное обеспечение

## Уровни логирования

Trace	Полная информация о состоянии программы или ее части.
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.



# Инструменты отладки ПО

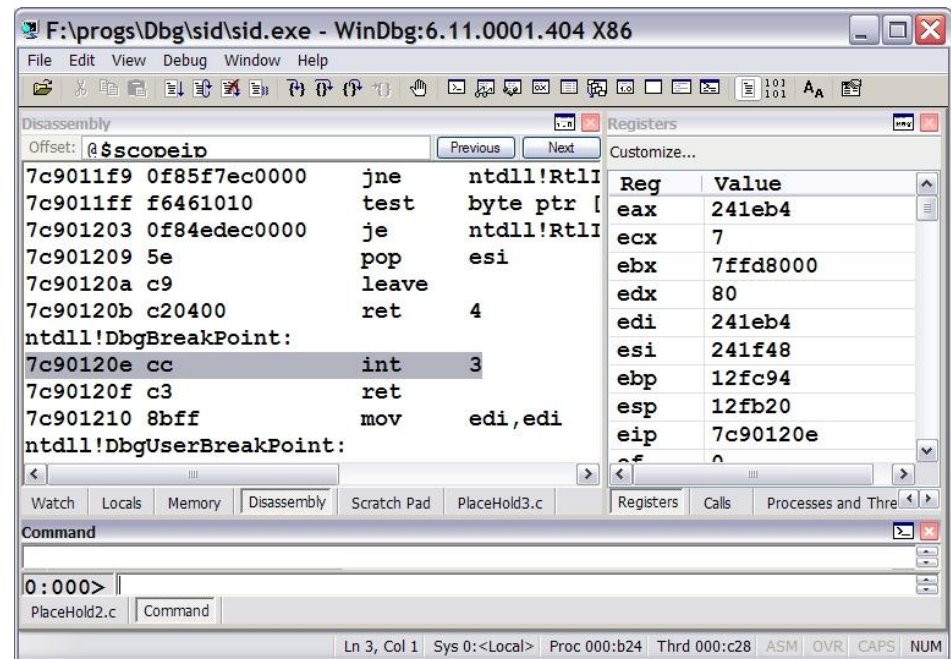
- Отладчик (Debugger)
- Профилировщик (Profiler)
- Детектор утечек памяти (Leak Detector)
- Статический анализатор кода (Static Analyzer)
- Динамический анализатор кода (Dynamic Analyzer)



# Инструменты отладки ПО: Отладчик (Debugger)

Возможности:

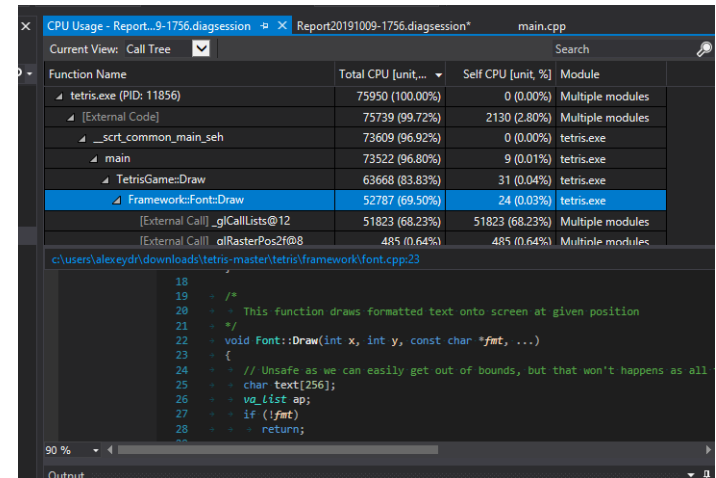
- Пошаговое исполнение программы
- Отслеживание и изменение значений переменных
- Настройка условий остановки



# Инструменты отладки ПО: Профилировщик (Profiler)

Профилировщик определяет, как долго и как часто выполняется код:

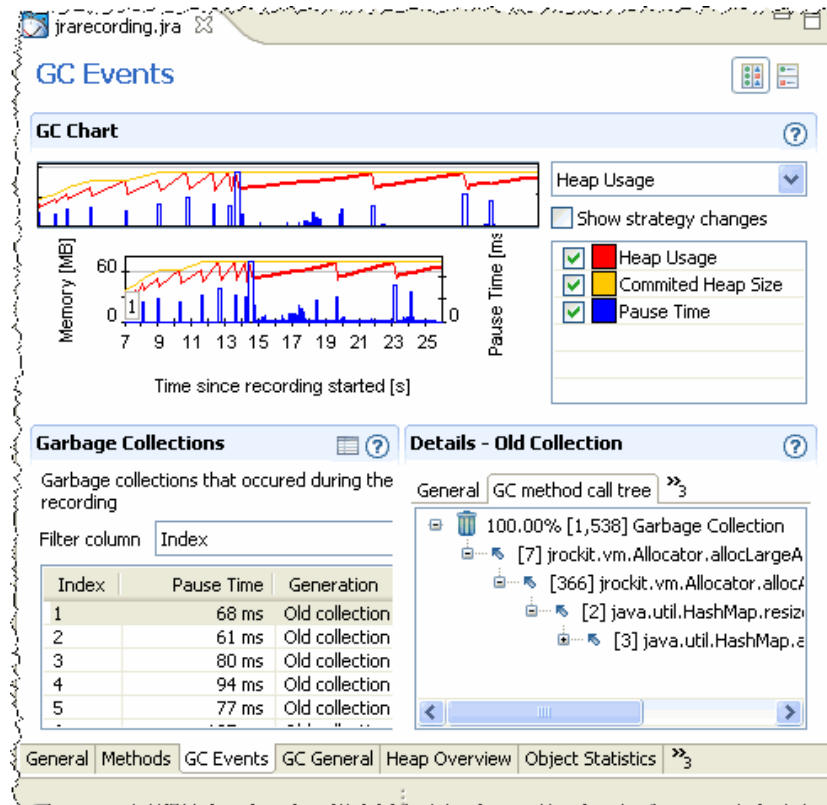
- Анализ отдельных функций и строк кода.
- Анализ конкретного временного диапазона.
- Анализ конкретного потока программы.
- Определяет количество вызовов и время, которое проводится внутри функции.
- Время может включать или исключать время вложенных вызовов.



# Инструменты отладки ПО: Детектор утечек памяти

Возможности:

- Выявление “утечки” памяти
- Анализ использования и утечек памяти
- Анализ использования и утечек иных ресурсов

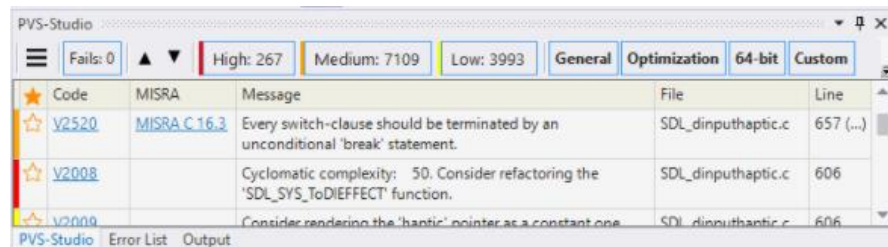




# Инструменты отладки ПО: Анализатор кода

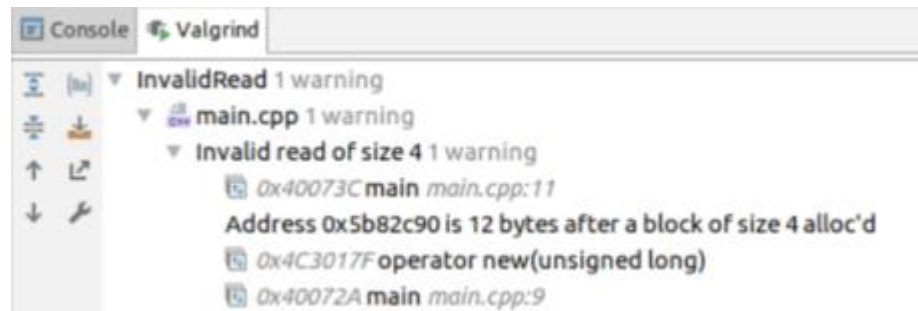
Статический анализатор:

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



Динамический анализатор:

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







# Тестирование

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



# 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

```
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
```

## New Function Breakpoint

Break on any function matching the specified function name that is identified while debugging

Function Name:

Language: C++

☐ Conditions

☐ Actions

OK

Cancel

## New Data Breakpoint

Break execution when the specified number of Bytes are modified starting at the specified Address

Address: '8&file' (4 Bytes)

☐ Conditions

☒ Actions

Log a message to Output Window:  [?](#) Cancel

☒ Continue execution

OK

Cancel

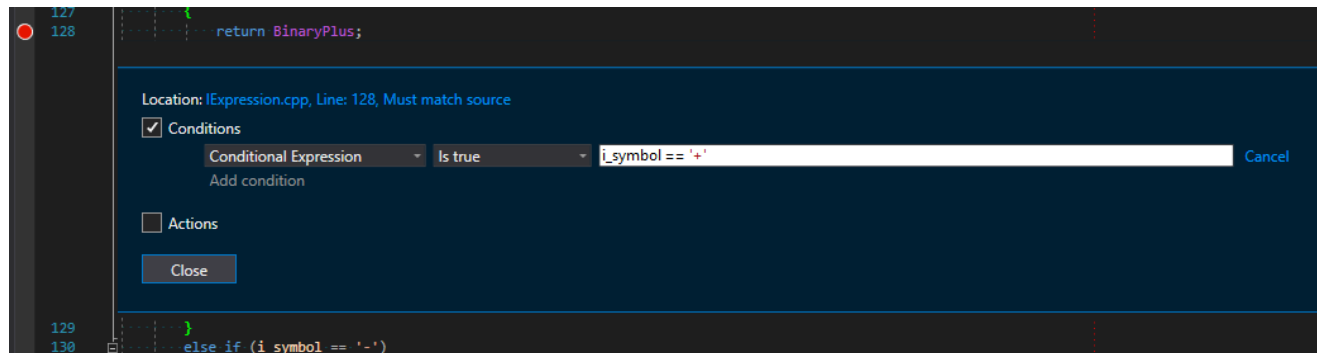
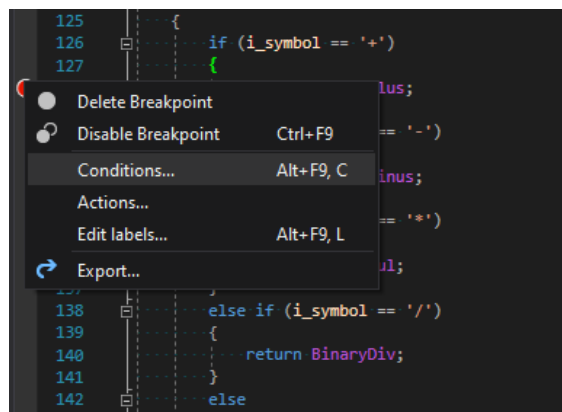


# Отладчик Visual Studio: Точки останова: основные типы

- Simple Breakpoint – остановка программы при выполнении конкретной строки кода.
- Function Breakpoint – остановка программы при вызове функции с указанным именем. Удобно использовать для того, чтобы остановить программу при вызове любой функции класса.
- Data Breakpoint – остановка программы при изменении памяти по указанному адресу. Обычно ограничено размером указателя (4 или 8 байт).



# Отладчик Visual Studio: Точки останова: настройка



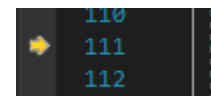
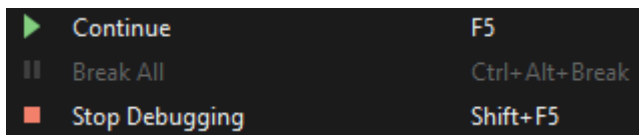


# Отладчик Visual Studio: Точки останова: настройка

- Conditions: Conditional Expression – остановка программы только при выполнении указанного условия.
- Conditions: Hit Count – остановка программы только при определенном количестве попаданий в точку останова.
- Conditions: Filter – остановка программы только при выполнении дополнительных критериев (например, только в конкретном потоке).
- Actions – напечатать сообщение в окно Output и, опционально, продолжить работу.

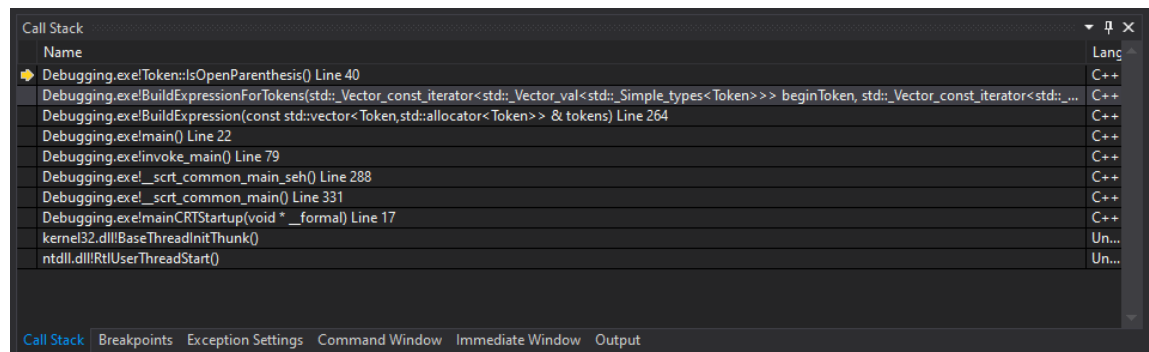
# Отладчик Visual Studio: Выполнение программы

- 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 строчки
- Set Next Statement – нарушить поток исполнения и немедленно перейти в выбранную точку программы



# Отладчик Visual Studio: Стек вызовов

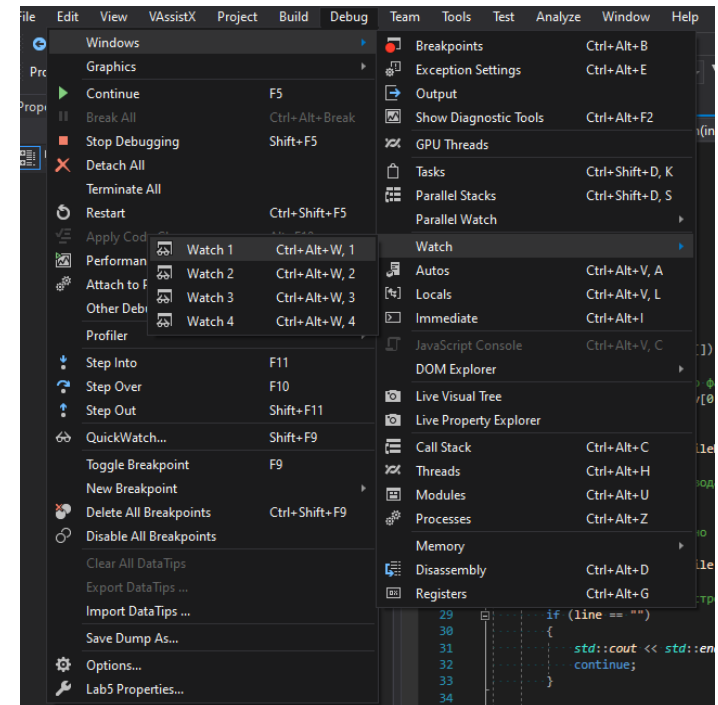
- Показывает полную цепочку вызовов функций, которая привела к текущему состоянию программы
- Сверху – текущая точка исполнения программы
- Снизу – точка входа main()





# Отладчик Visual Studio: Просмотр переменных

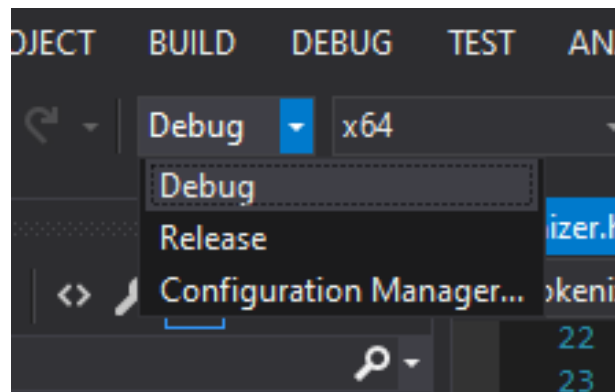
- Просмотр значений глобальных и локальных переменных
- Просмотр значений простых выражений
- Изменение значений переменных
- Форматирование значений в удобной форме



Name	Value	Type
file	0x00000000 <NULL>	_ioobuf *
d	2.2260063532870538e-201	double
pi	5.7417353307295620e+285	const double
eax	0x7b435750	unsigned int
esp	0x0082fb10	unsigned int
first	{x=0x772fea4b y=0x00000002 }	main::_l2::point
x	0x772fea4b	int
y	0x00000002	int
&first	0x0082faa4 {x=0x772fea4b y=0x00000002 }	main::_l2::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, но отладка при этом может быть затруднена или вовсе невозможна.





# Crash dump

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

# Crash dump: Назначение

Содержит информацию о состоянии программы в определённый момент времени:

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





# Crash dump: Создание (вариант 1)

```
#include <dbgghelp.h>
```

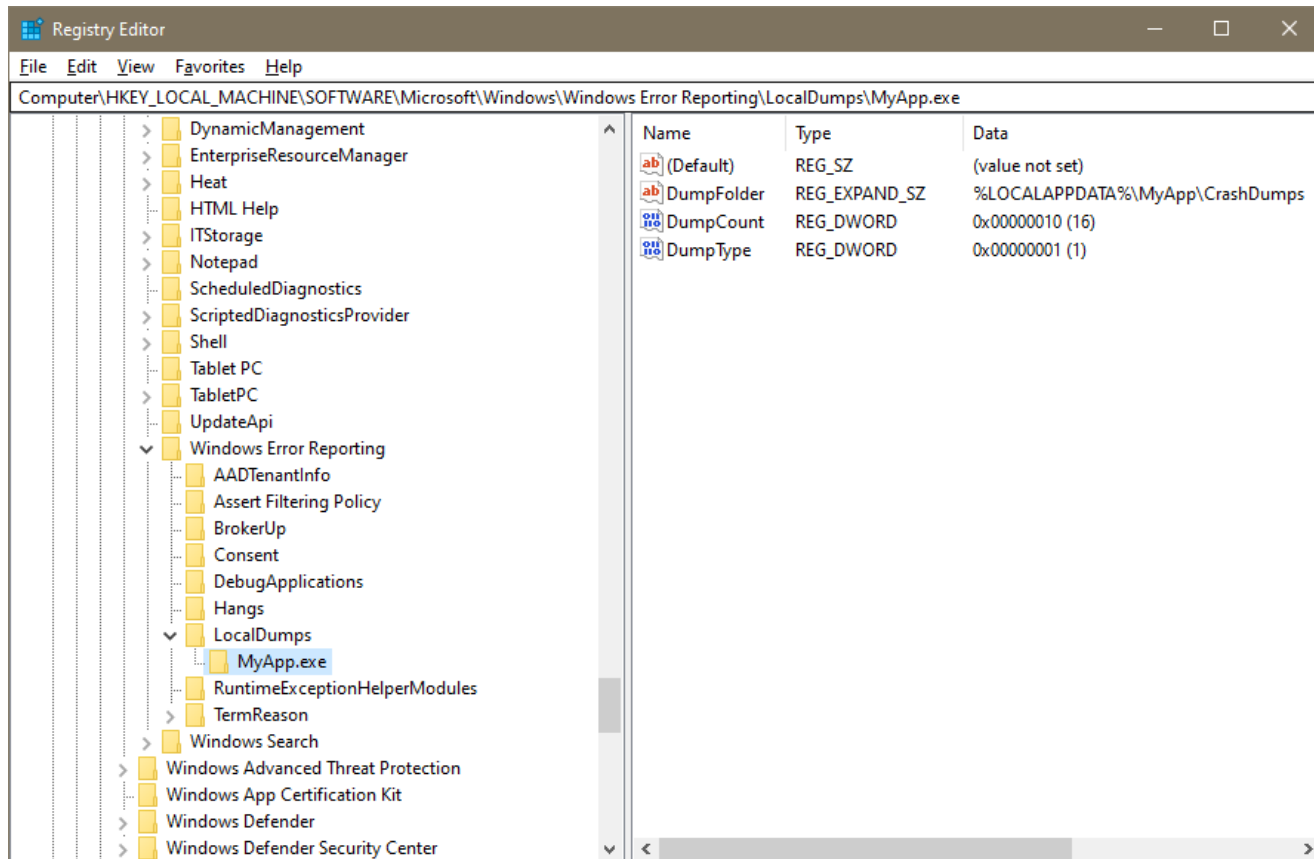
```

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 );

```

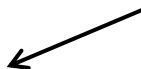
```
#include <windows.h>
LONG CustomTopLevelFilter(_EXCEPTION_POINTERS *pExceptionInfo )
{
    return GenerateDump(pExceptionInfo);
}
void SetExceptionHook()
{
    ::SetUnhandledExceptionFilter(TopLevelFilter );
}
```

# Crash dump: Создание (вариант 2)



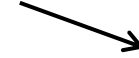
# Crash dump: Анализ

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 12:30:58.000 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 .ecxr.
(a781900): Integer divide-by-zero - code c0000004 (first/second chance not available)
eax=00000000 ebx=00380d78 ecx=00000000 edx=00000000 esi=00380d38 edi=0017ebd8
eip=77e40c22 esp=0017e898 ebp=0017e8a8 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  fs=002b  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 12:08:38 PM

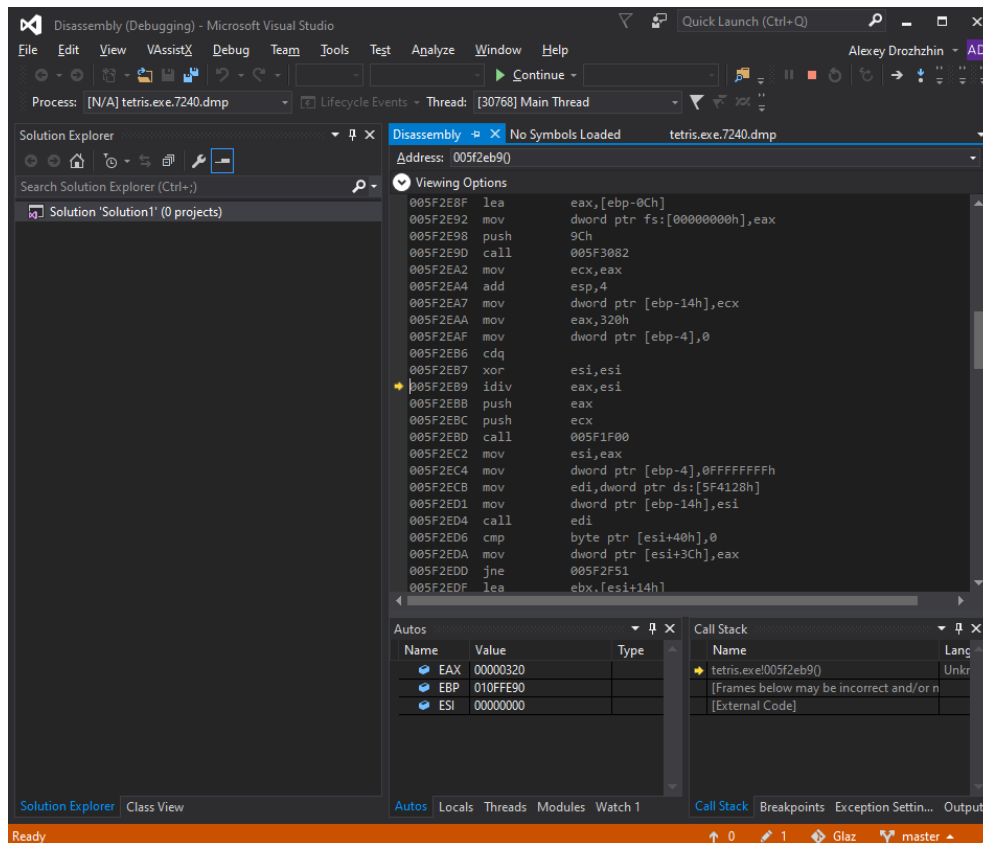
Dump Summary		
Dump File	tetris.exe.7240.dmp	D:\dumps\tetris\CrashDumps\tetris.exe.7240.dmp
Last Write Time	10/11/2019 12:08:38 PM	
Process Name	tetris.exe	C:\Users\alexeid\Downloads\tetris-master\Release\tetris.exe
Process Architecture	x86	
Exception Code	0xC0000094	
Exception Information	The thread tried to divide an integer value by an integer divisor of zero.	
Heap Information	Not Present	
Error Information		

System Information		
OS Version	10.0.18362	
CLR Version(s)		

Modules		
Module Name	Module Version	Module Path
tetris.exe	0.0.0.0	C:\Users\alexeid\Downloads\tetris-master\Release\tetris.exe
ntdll.dll	10.0.18362.1	C:\Windows\System32\ntdll.dll
kernel32.dll	10.0.18362.86	C:\Windows\System32\kernel32.dll

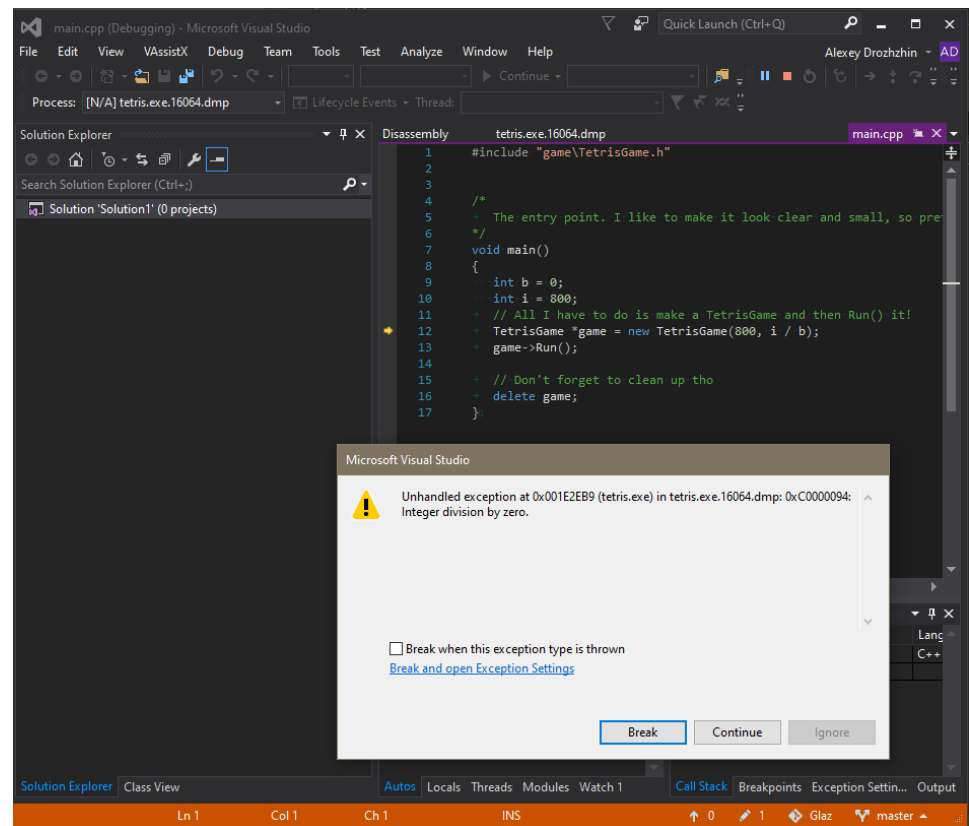
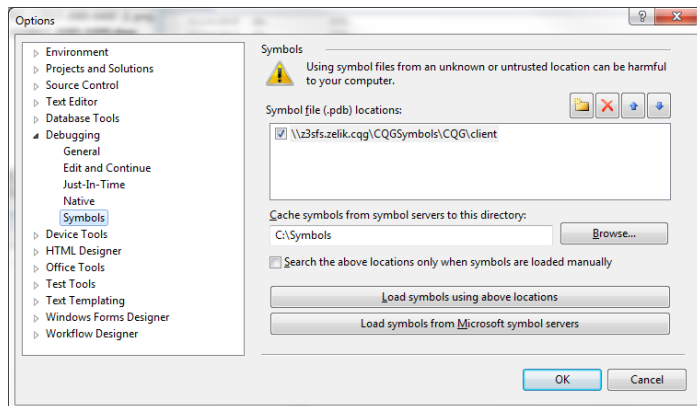
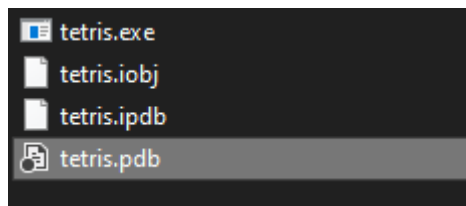
Output  
Show output from: General  
We were unable to automatically populate your Visual Studio Team Services accounts.  
The following error was encountered: VS30063: You are not authorized to access <https://app.vssps.visualstudio.com>.

# Crash dump: Анализ (без PDB)






# Crash dump: Анализ (с PDB)





# Типовые ошибки

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

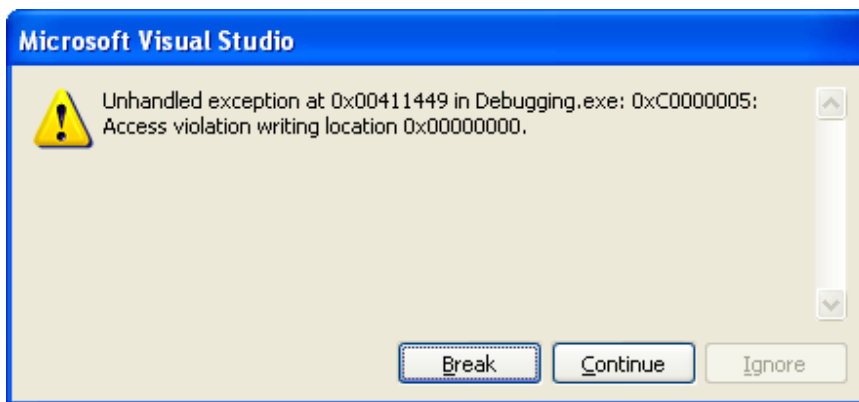


# Типовые ошибки: Общие закономерности

- 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.
- Но это не ваш случай.

# Типовые ошибки

- Access violation
- Memory leak
- Heap corruption
- Stack overflow



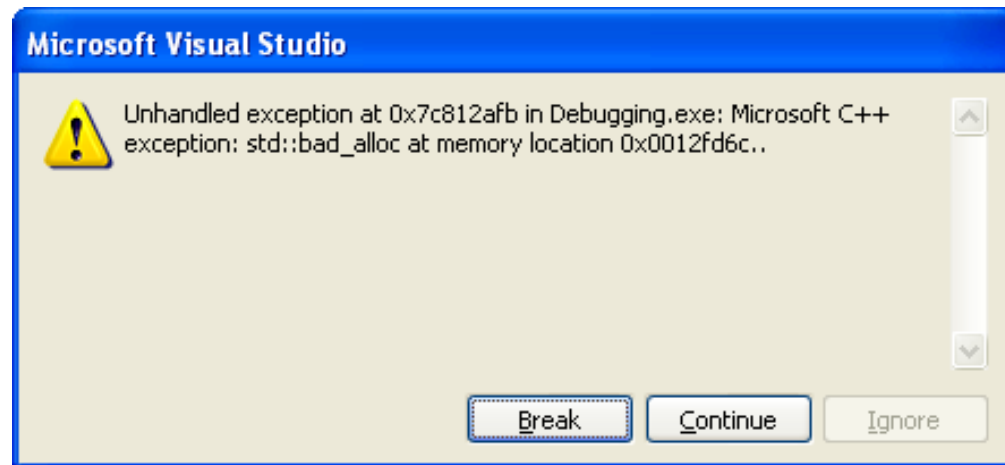
## Причины:

- Неинициализированные переменные и члены класса
- Осиротевшие указатели



# Типовые ошибки

- Access violation
- Memory leak
- Heap corruption
- Stack overflow

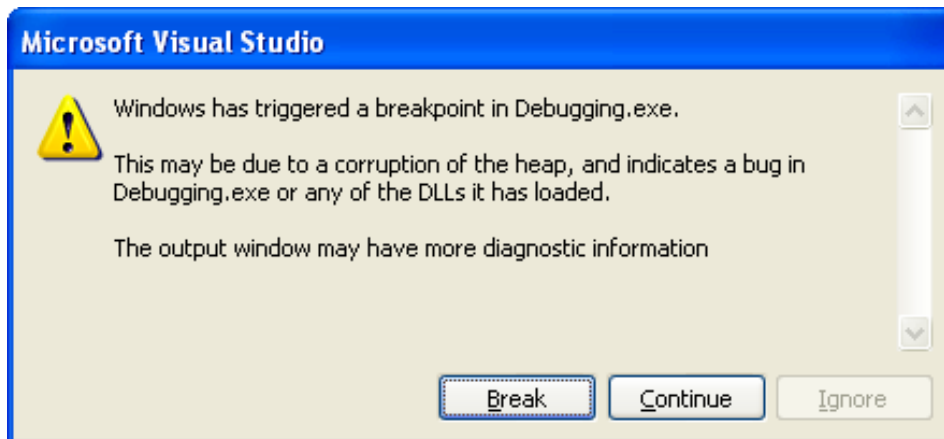


## Причины:

- Забытый delete
- Не виртуальный деструктор в базовом классе

# Типовые ошибки

- Access violation
- Memory leak
- Heap corruption
- Stack overflow

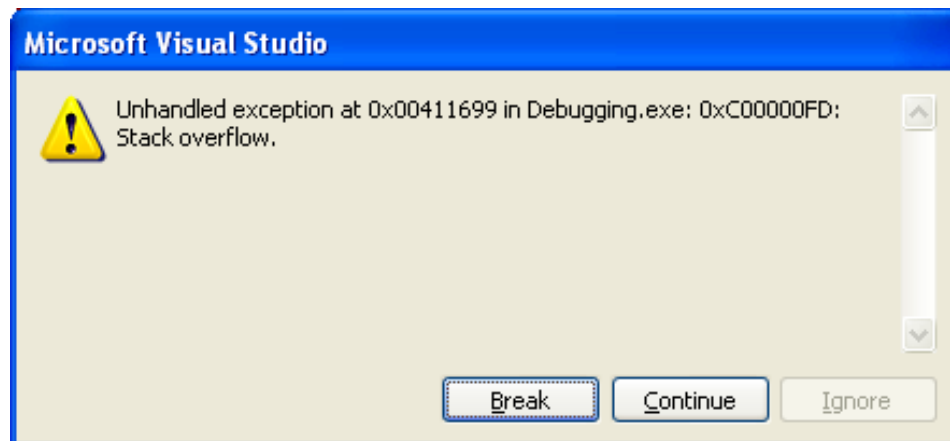


## Причины:

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

# Типовые ошибки

- Access violation
- Memory leak
- Heap corruption
- Stack overflow



Причины:

- “Бесконечная” рекурсия



# 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