Основы инженерии требований

Стандартизация и стандарты

По происхождению программные продукты бывают двух типов: заказные (под заказ конкретного потребителя) и коробочные (для массовой продажи на рынке). Для заключения контракта заказчик должен быть уверен, что разработчик справится и не завалит проект. Вопрос: как его в этом убедить?

В мировой практике промышленного производства ответы на эти вопросы дают стандарты на производство продуктов и услуг и сертификация производителей на соответствие этим стандартам.

Профессиональный стандарт

SWEBOK (Стандарт ISO/IEC TR 19759 IEEE, 15.09.2005) – дает представление о знаниях программного инженера, имеющего степень бакалавра и четырехлетний опыт работы

SWEBOK (SoftWare Engineering Body Of Knowledge)

Содержит описания состава знаний по следующим 10 разделам (областям знаний) программной инженерии:

Software Requirements – требования к ПО

Software Design – проектирование ПО

Software Construction – конструирование ПО

Software Testing – тестирование ПО

Software Maintenance – сопровождение ПО

SWEBOK (SoftWare Engineering Body Of Knowledge)

Software Configuration Management – управление конфигурациями

Software Engineering Management – управление IT проектом

Software Engineering Process – процесс программной инженерии

Software Engineering Tools and Methods – методы и инструменты

Software Quality – качество ПО

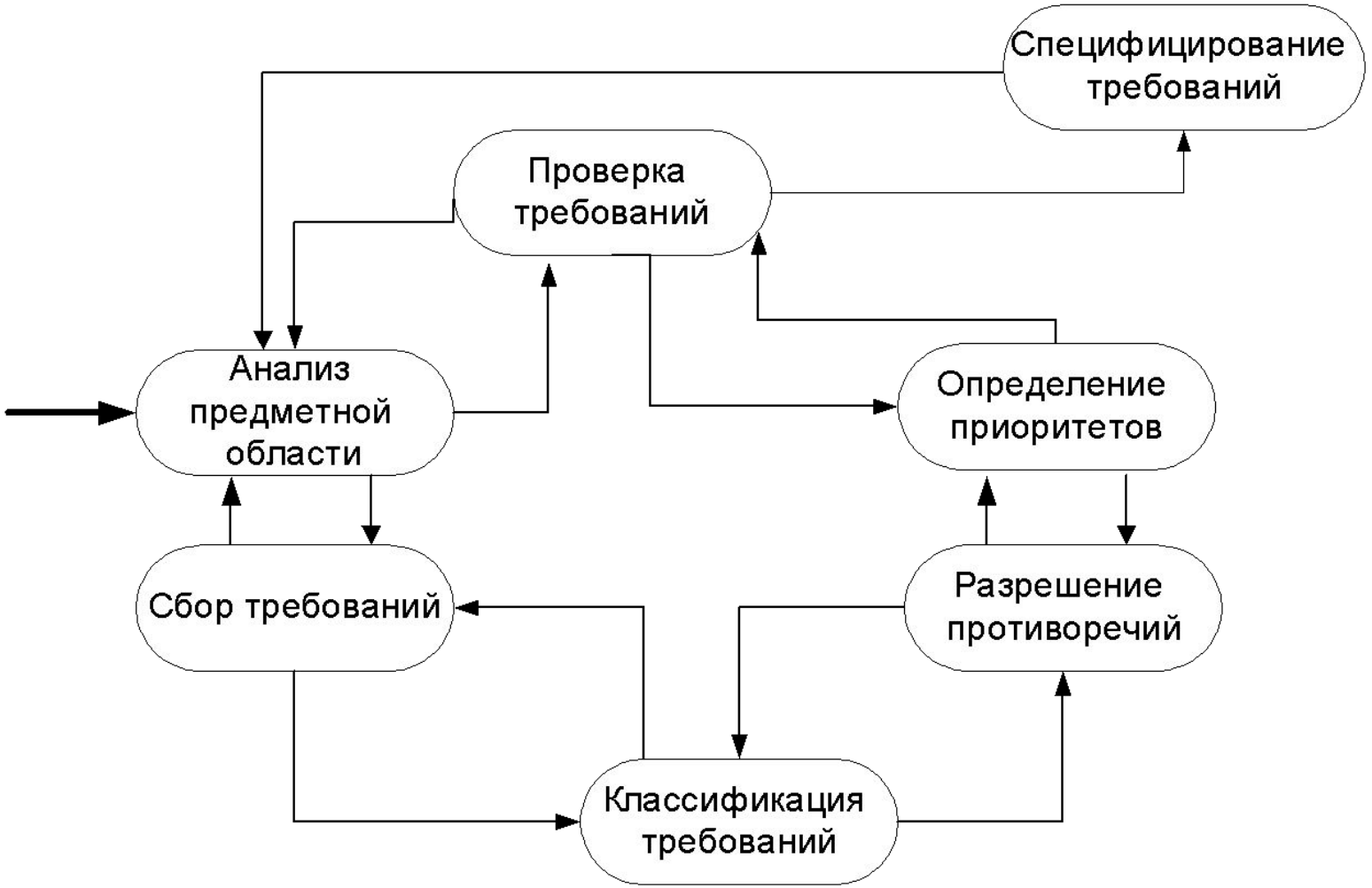
Подробнее: Guide to the Software Engineering Body of Knowledge - http://www.swebok.org/

Определение требований к программной системе

Существуют функциональные и нефункциональные требования. Функциональные требования регламентируют функционирование или поведение системы, они должны отвечать на вопрос: «что должна делать система», абстрагируясь от вопроса «как она это должна делать».

Нефункциональные требования, соответственно, регламентируют внутренние и внешние условия или атрибуты функционирования системы (примеры: ограничения, атрибуты качества, внешние интерфейсы).

Общая модель формирования и анализа требований



Тестирование программ

Тестирование программ

- **Тестирование** – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением в специально заданных, искусственных условиях.

Виды тестирования

- **Модульное тестирование** - тестируется отдельный модуль, в отрыве от остальной системы. Самый распространенный случай применения – тестирования модуля самим разработчиком, проверка того, что отдельные модули, классы, методы делают действительно то, что от них ожидается.

Виды тестирования

- **Интеграционное тестирование** – два и более компонентов тестируются на совместимость. Это очень важный вид тестирования, поскольку разные компоненты могут создаваться разными людьми, в разное время, на разных технологиях. Этот вид тестирования, безусловно, должен применяться самими программистами, чтобы, как минимум, удостовериться, что все живет вместе в первом приближении..

Виды тестирования

- **Системное тестирование** – это тестирование всей системы в целом, как правило, через ее пользовательский интерфейс. При этом тестировщики, менеджеры и разработчики акцентируются на том, как ПО выглядит и работает в целом, удобно ли оно, удовлетворяет ли она ожиданиям заказчика. При этом могут открываться различные дефекты, такие как неудобство в использовании тех или иных функций, забытые или "скудно" понятые требования.

Виды тестирования

- **Нагрузочное тестирование** – тестирование системы на корректную работу с большими объемами данных. Например, проверка баз данных на корректную обработку большого (предельного) объема записей, исследование поведение серверного ПО при большом количестве клиентских соединений, эксперименты с предельным трафиком для сетевых и телекоммуникационных систем, одновременное открытие большого числа файлов, проектов и т.д.

Тестирование по принципу «черного ящика»

- Под **«чёрным ящиком»** понимается объект исследования, внутреннее устройство которого неизвестно.
- Тестировщик имеет: требования к системе, описывающие ее поведение, и саму систему, работать с которой он может, только подавая на ее входы некоторые внешние воздействия и наблюдая на выходах некоторый результат. Все внутренние особенности реализации системы скрыты от тестировщика.

Тестирование по принципу «черного ящика»

- Основная задача тестировщика для данного метода тестирования состоит в последовательной проверке **соответствия поведения системы требованиям**. Кроме того, тестировщик должен проверить **работу системы в критических ситуациях** - что происходит в случае подачи неверных входных значений.

Тестирование по принципу «черного ящика». Пример

- Работаем с программой, которая реализует поиск в заданном массиве серии нулевых элементов минимальной длины.
- Тесты «черного ящика»:
  - по входным данным: некорректная длина массива, нечисловые значения в массиве.
  - по работе основного алгоритма: массив без нулевых элементов; массив, в котором ТОЛЬКО нулевые элементы; массив, содержащий несколько серий нулевых элементов минимальной длины; массив, содержащий несколько серий максимальной длины и т.д.

Эквивалентное разбиение (метод «черного ящика»)

- Основу метода составляют два положения:
  - Исходные данные необходимо разбить на **конечное число классов эквивалентности**. В одном классе эквивалентности содержатся такие тесты, что, если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку.
  - Каждый тест должен включать, по возможности, **максимальное количество классов эквивалентности**, чтобы минимизировать общее число тестов.

Пример эквивалентного разбиения

Входное условие	Правильные классы эквивалентности	Неправильные классы экивалентности
Длина массива N -целое положительное число, >0	N – целое положительное число, >0	N – нецелое число; N – целое отрицательное число; N =0

Тесты для правильного класса:

-N=5, N=8, N=32000.

Тесты для неправильных классов:

-N-нецелое число: N=4.8, N=“str”, N=1234567889;

-N – целое отрицательное число: N=-345, N=-2, N=-1234;

-N=0: N=0.

Анализ граничных условий (метод «черного ящика»)

- В тестовых примерах, прямо соответствующих тест-требованиям, обычно используются входные значения, находящиеся заведомо **внутри допустимого диапазона**. Один из способов проверки устойчивости системы на значениях, близких к предельным, - создавать для каждого входа как минимум пять тестовых примеров:
  - Значение внутри диапазона
  - Минимальное значение
  - Максимальное значение
  - Минимальное значение -1
  - Максимальное значение +1

Пример анализа граничных условий

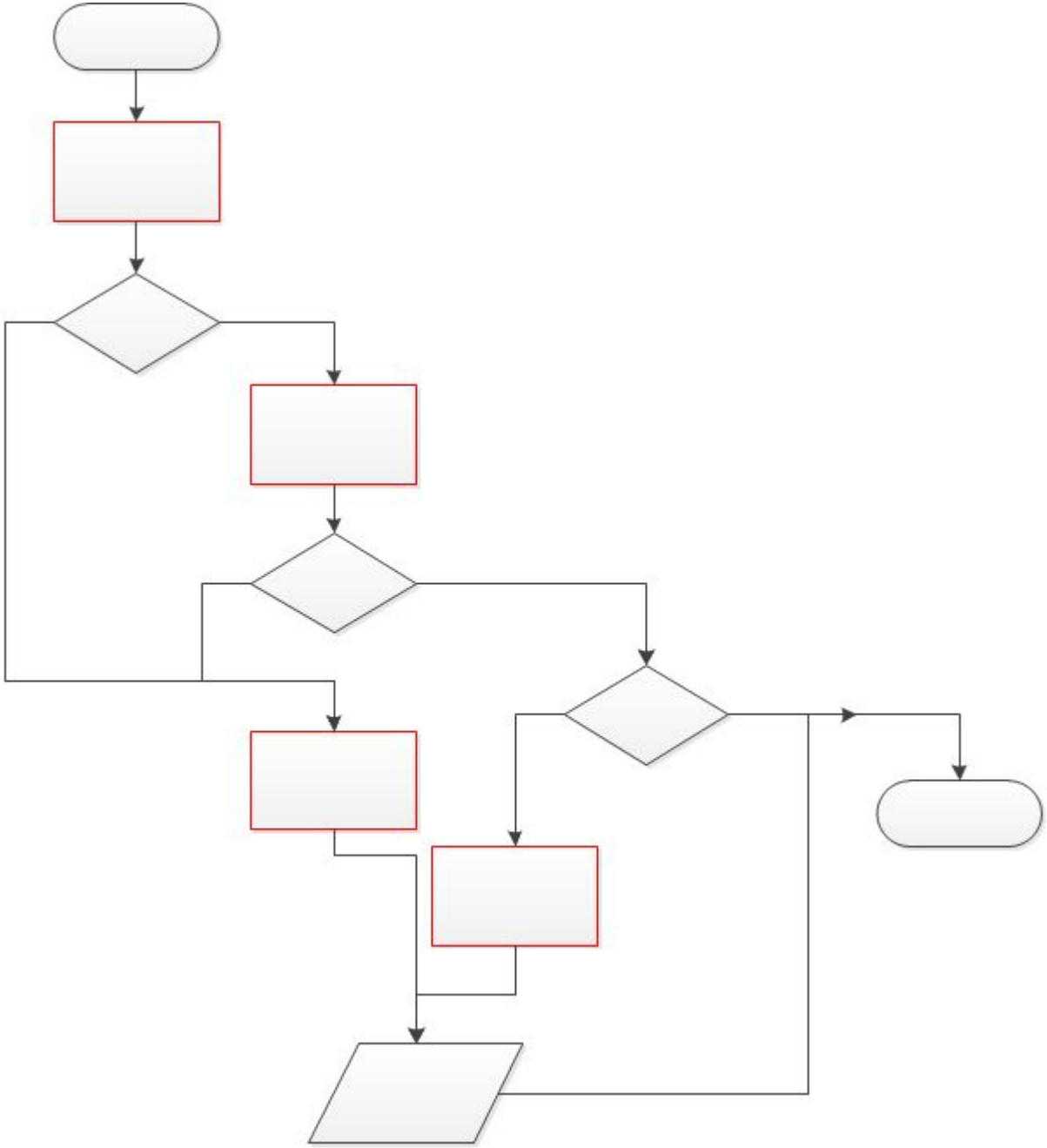
- N – длина массива (целое число), соответствующие пять тестов:
  - N=20, N=145, N=540;
  - N=1;
  - N= 32767;
  - N=0;
  - N= 32768.

Тестирование по принципу “белого ящика”

- При **тестировании методами “белого ящика”** код программ доступен тестировщикам и используется в качестве источника информации о работе программы.

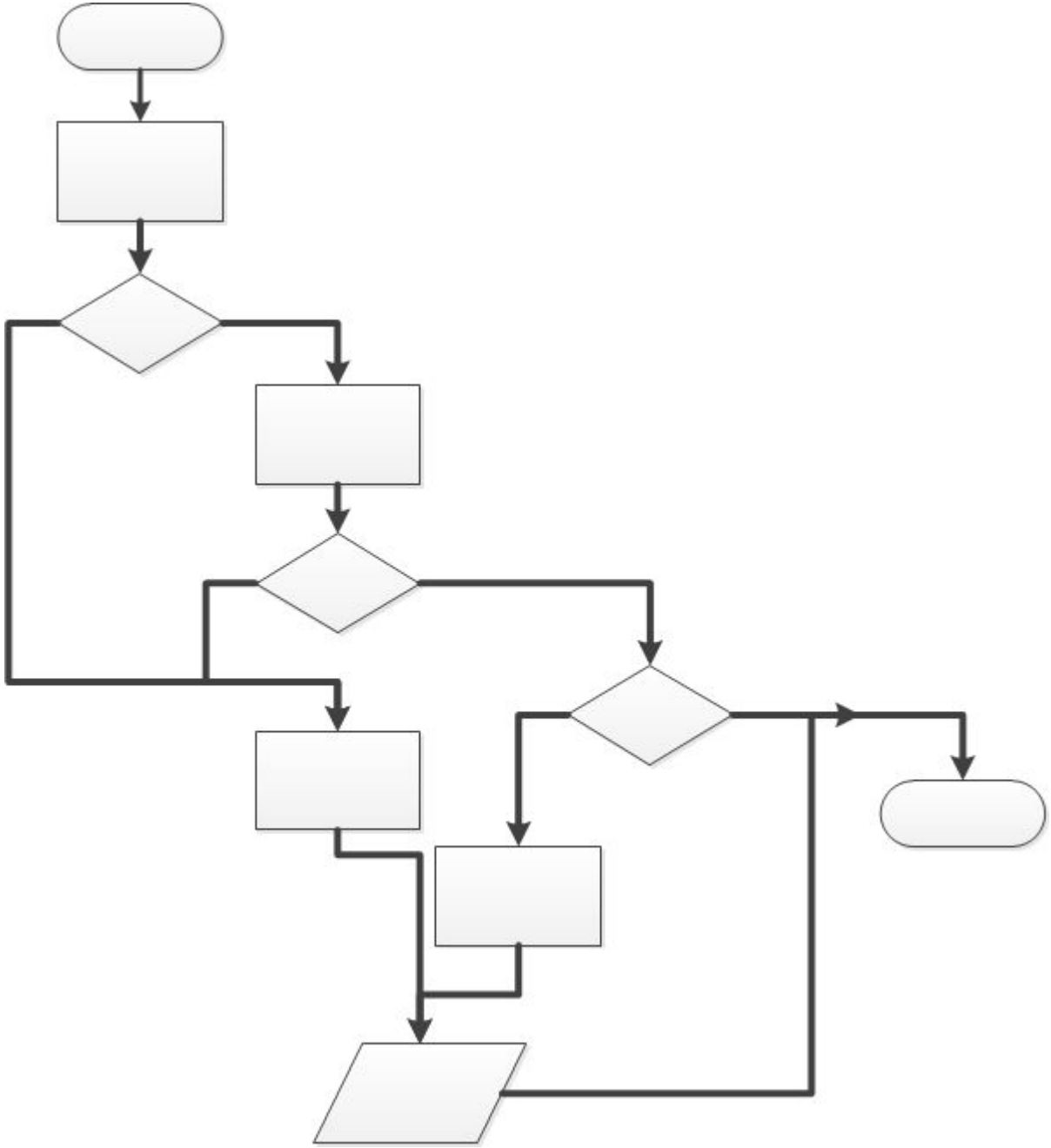
Метод покрытия операторов («белый ящик»)

- **Критерий покрытия операторов** подразумевает выполнение каждого оператора программы хотя бы один раз. Это необходимое, но не достаточное условие для приемлемого тестирования.



Метод покрытия переходов

- **Критерий покрытия переходов** подразумевает поиск тестов для прохода по каждому переходу хотя бы один раз. Это также необходимое, но не достаточное условие.



Отладка программ

Отладка программы — это специальный этап в разработке программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен. Программные ошибки, как правило, делятся на три вида:

- ***Синтаксическая ошибка***. Неправильное употребление синтаксических конструкций, например употребление оператора цикла For без то или Next.

- ***Семантическая ошибка***. Нарушение семантики той или иной конструкции, например передача функции параметров, не соответствующих ее аргументам.
- ***Логическая ошибка*** . Нарушение логики программы, приводящее к неверному результату. Это наиболее трудный для "отлова" тип ошибки, ибо подобного рода ошибки, как правило, кроются в алгоритмах и требуют тщательного анализа и всестороннего тестирования.

В современных средах разработки существуют следующие инструменты отладки:

-Точки останова;

-Пошаговое выполнение;

-Контрольные значения.

Точки останова программы

При отладке широко используется метод, который называют методом точек останова.

В программировании, **точка останóва** (*breakpoint*) — это преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика (одновременно с этим, программа сама может использовать точки останова для своих нужд)

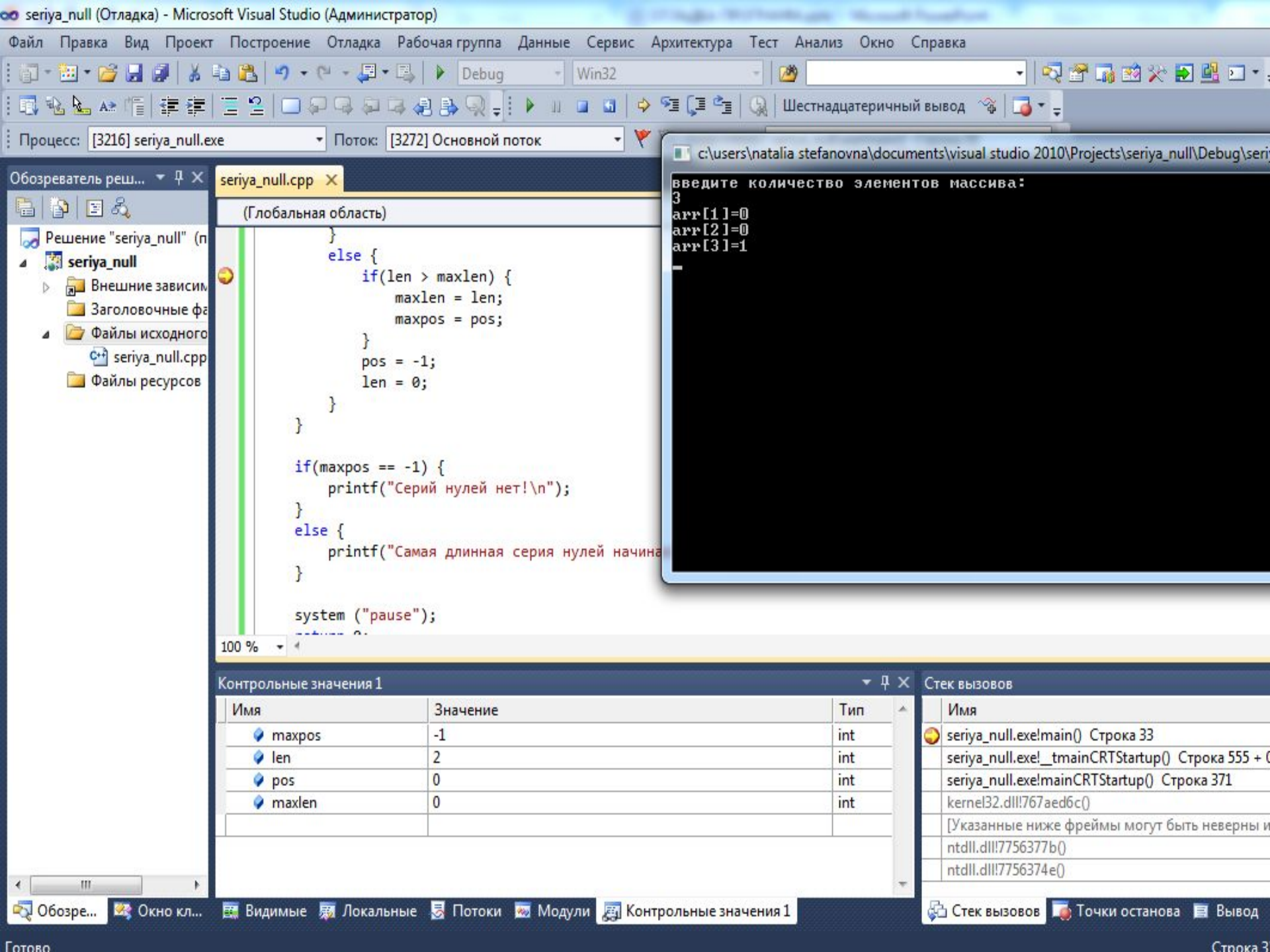
Суть метода заключается в том, что программист помечает некоторые инструкции программы (ставит точки останова), при достижении которых программа приостанавливает свою работу, и программист может начать трассировку или проконтролировать значения переменных.

Задача. В одномерном массиве найти серию нулей и посчитать количество элементов в серии

```
#include <stdio.h>
#include <windows.h>
#include <locale.h>
#define MAX_N 100

int main() {
    setlocale(LC_ALL, "rus");
    int n;
    printf ("введите количество элементов массива:\n");
    do {
        scanf("%d",&n);
    } while (n<=0 || n>MAX_N);

    int arr[MAX_N];
```

```
for (int i=0;i<n;i++) {
    printf("arr[%d]=", i+1);
    scanf("%d", &arr[i]);
}

int pos = -1, len = 0;
int maxpos = -1, maxlen = 0;

for(int i = 0; i < n; i++) {
    if(arr[i] == 0) {
        if(pos == -1) {
            pos = i;
        }
    len++;
}
```

```
            else {
                if(len > maxlen) {
                    maxlen = len;
                    maxpos = pos;
                }
                pos = -1;
                len = 0;
            }
        }

        if(maxpos == -1) {
            printf("Серий нулей нет!\n");
        } else {
            printf("Самая длинная серия нулей начинается с позиции %d и равна
%d",
                    maxpos+1, maxlen);
        }

        system ("pause");
        return 0;
}
```


seriya_null (Отладка) - Microsoft Visual Studio (Администратор)
Файл Правка Вид Проект Построение Отладка Рабочая группа Данные Сервис Архитектура Тест Анализ Окно Справка
Debug
Win32
Шестнадцатеричный вывод
Процесс: [3216] seriya_null.exe
Поток: [3272] Основной поток
c:\users\natalia stefanovna\documents\visual studio 2010\Projects\seriya_null\Debug\seri
введите количество элементов массива:
3
arr[1]=0
arr[2]=0
arr[3]=1
Обозреватель реш...
Решение "seriya_null" (п
seriya_null
Внешние зависим
Заголовочные фа
Файлы исходного
seriya_null.cpp
Файлы ресурсов
seriya_null.cpp
(Глобальная область)
}
else {
if(len > maxlen) {
maxlen = len;
maxpos = pos;
}
pos = -1;
len = 0;
}
}
if(maxpos == -1) {
printf("Серий нулей нет!\n");
}
else {
printf("Самая длинная серия нулей начина
}
system ("pause");
100 %
Контрольные значения 1
Имя
Значение
Тип
maxpos
-1
int
len
2
int
pos
0
int
maxlen
0
int
Стек вызовов
Имя
seriya_null.exe!main() Строка 33
seriya_null.exe!__tmainCRTStartup() Строка 555 + 0
seriya_null.exe!mainCRTStartup() Строка 371
kernel32.dll!767aed6c()
[Указанные ниже фреймы могут быть неверны и
ntdll.dll!7756377b()
ntdll.dll!7756374e()
Обозре...
Окно кл...
Видимые
Локальные
Потоки
Модули
Контрольные значения 1
Стек вызовов
Точки останова
Вывод
Готово
Строка 3

Контрольные значения

Контрольные значения – это набор переменных/полей/свойств, значения которых наблюдаются при отладке (т.е. при пошаговом выполнении программы).

Пошаговое выполнение

Одной из наиболее распространенных процедур отладки является *пошаговое выполнение*.

При пошаговом выполнении код выполняется по одной строке за раз.

Во время остановки выполнения, например при достижении отладчиком точки останова, можно осуществлять пошаговое выполнение кода с помощью трех команд меню **Отладка**.

Пошаговое выполнение

Команда меню	Сочетание клавиш	Описание
Шаг с заходом	**F11**	Если строка содержит вызов функции, команда **Шаг с заходом** выполняет только сам вызов, а затем останавливает выполнение в первой строке кода внутри функции. В противном случае команда **Шаг с заходом** выполняет следующий оператор.
Шаг с обходом	**F10**	Если строка содержит вызов функции, команда **Шаг с обходом** выполняет вызываемую функцию, а затем останавливает выполнение в первой строке кода внутри вызывающей функции. В противном случае команда **Шаг с заходом** выполняет следующий оператор.
Шаг с выходом	**Shift+F11**	Команду **Шаг с выходом** возобновляет выполнение кода до возврата функции, а затем прерывает выполнение в точке возврата вызывающей функции.

Пошаговое выполнение

Шаг отладки - это переход с текущей строки, на которой остановилась программа, на следующую.

Отладчик фактически осуществляет пошаговое выполнение операторов кода, а не физических строк.

Пошаговое выполнение

- если мы используем шаг с обходом, то мы просто переходим по коду на след строку;
- если используем шаг с заходом, то, если текущая строка - это вызов функции, мы переходим на 1 строку этой функции (заходим в функцию);
- если мы находимся внутри функции (какой-то своей, или main) и используем шаг с выходом, мы возвращаемся "вверх", т.е. к месту, где эта функция была вызвана, и переходим к след за вызовом строке;

Отладка приложения в Visual Studio

Существуют два способа запуска режима отладки в Visual Studio:

-запуск из меню Debug;

-запуск с помощью соответствующей панели инструментов.

Оба способа - предоставляют доступ к запуску сеансов отладки, пошаговому прохождению кода, управлению точками останова, а также и ко многим функциональным возможностям отладки в Visual Studio.

Имеются два состояния меню отладки (Debug):

-Состояние покоя (неактивное);

-Режим отладки.

Рассмотрим меню Debug в режиме отладки. Когда отладчик запущен, то состояние меню Debug изменяется, в нем становятся активными несколько дополнительных опции. Эти опции включают: функции перемещения по коду, перезапуск сеанса и доступ к дополнительным окнам отладки и т.д.

Элементы меню Debug в режиме отладки

Windows | Break points Позволяет открыть окно Breakpoints во время сеанса отладки.

Windows | Output Открывает окно Output во время активного сеанса отладки для того, чтобы можно было читать выходные сообщения, выдаваемые компилятором и отладчиком.

Элементы меню Debug в режиме отладки

Windows | Watch Открывает одно из нескольких окон контрольных значений интегриро ванной среды. Окна контрольных значений представляют элементы и выражения, за которыми вы наблюдаете в течение сеанса отладки.

Windows | Autos Открывает окно Autos. Это окно показывает переменные (и их значе ния) в текущей и предыдущей строках кода.

Элементы меню Debug в режиме отладки

Windows | Locals Открывает в интегрированной среде окно Locals, которое показывает переменные в локальной области действия (функции).

Windows | Immediate Открывает окно Immediate, в котором вы можете выполнить команду.

Элементы меню Debug в режиме отладки

Windows | Call Stack Открывает список функций, которые имеются в стеке. Также указывает текущий кадр стека (функцию). Выделенный элемент - это то, что определяет содержимое окон Locals. Autos и окон контрольных значений.

Windows | Registers Открывает окно Registers, чтобы вы могли видеть изменения значений регистров при прохождении по коду. Данная функция работает только при включенной (в диалоговом окне Options) отладке на уровне адресов.

Элементы меню Debug в режиме отладки

Continue Продолжает выполнение приложения после выхода в интегрированную среду разработки. Приложение продолжает выполняться с активной строки кода.

Break All Позволяет прервать приложение вручную (без использования точки останова) во время сеанса отладки Приложение прервется на сле дующей исполняемой строке. Эта возможность полезна в том случае, когда нужно получить доступ к

Элементы меню Debug в режиме отладки

Stop Debugging Останавливает режим отладки. Прерывает также и отлаживаемый процесс.

Restart Останавливает и перезапускает его сеанс отладки. Аналогично последовательному нажатию кнопок Stop Debugging и Start Debugging.

Элементы меню Debug в режиме отладки

Exceptions Активирует диалог Exceptions, который позволяет управлять выходом в IDE по конкретным типам исключительных состояний .NET Framework и других библиотек.

Step Into Приводит к продвижению отладчика на одну строку.

Step Over Работает точно так же, как Step Into, но с одной важной разницей: если используете пропуск функции по Step Over, то строка вызова функции будет выполнена (и функция тоже), и отладчик установит следующую строку за вызовом функции в качестве следующей отлаживаемой строки.

Элементы меню Debug в режиме отладки

Step Out Указывает отладчику выполнить текущую функцию, а затем выйти на зад в отладчик (после выполнения функции). Эта функция полезна тогда, когда осуществляется вход в функцию, а затем нужно, чтобы эта функция выполнилась и вернула в отладчик после ее завершения.

QuickWatch Активирует окно QuickWatch. Это окно отображает значение одной, конкретной переменной (или выражения).

Элементы меню Debug в режиме отладки

Toggle Breakpoint Включает и выключает активную точку останова.

New Breakpoint Активирует диалоговое окно New Breakpoint.

Delete All Breakpoints Удаляет все точки останова в вашем решении

Установка точки останова

Приостановка выполнения программы в точке останова называется **режимом приостановки**.

Вход в режим приостановки выполнения не приводит к прекращению или завершению работы программы, поэтому выполнение программы может быть продолжено в любое время.

В Visual Studio можно помещать на любую строку кода, которая выполняется.

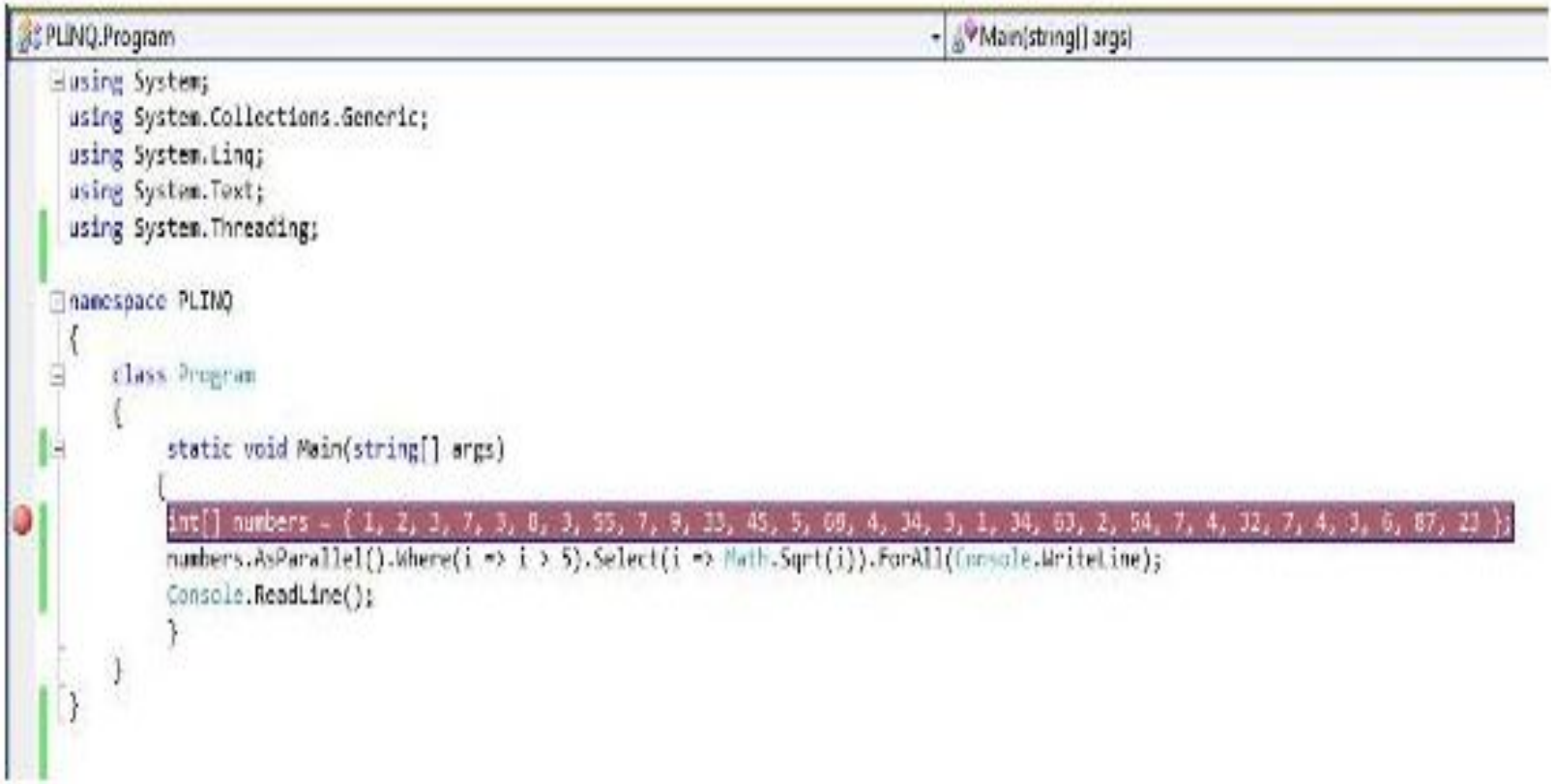
Существуют три способа расстановки точек останова в Visual Studio 2010:

-С помощью клавиши F9;

-Через пункт меню Debug - Toggle Breakpoint;

-И самый простой способ - это щелкнуть дважды левой кнопкой мыши на нужной строке, в окне редактора кода внутри затененной области вдоль левого края окна документа.

Точка останова обозначается большим кружком слева от соответствующей строки в окне редактора кода.



Run to Cursor (Ctrl + F10) (Выполнить до курсора)

Установите курсор на строке кода до которой вы хотите выполнить приложение, а затем нажмите вместе Ctrl + F10. Это приведет к выполнению приложения до этой строки и переходу в режим отладки сохраняя время которое могло быть потрачено на множественные нажатия F10/F11.

Это работает даже в тех случаях, когда код в который вы хотите попасть находится в отдельном методе или классе по отношению к тому месту которое вы в данный момент отлаживаете.

Conditional Breakpoints (Условные точки останова)

Условные точки останова позволяют вам переходить в режим отладки только, если какое-то определенное условие, которое было назначено, достигнуто.

Условные точки останова помогают избежать ручного изучения кода с его дальнейшим выполнением, а также могут сделать весь процесс отладки не

Как включить Conditional Breakpoint (Условную точку останова)

- Нажмите в коде F9, чтобы установить точку на определенной строке;
- Затем щелкните правой кнопкой мыши на красном кружке точки останова слева от окна редактора и выберите контекстное меню «Condition...» («Условие...»);

Это приведет к появлению диалогового окна, которое позволяет указать, что точка останова должна срабатывать только, если определенное условие истинно

Функция Hit Count (Число попаданий)

Иногда нужно, чтобы происходила остановка отладчика только при условии, что условие истинно N-раз.

Включить такое поведение можно правым щелчком на точке останова и выбором пункта меню «Hit count...» («Количество попаданий...»).

Это приведет к появлению диалогового окна, которое позволяет указать, что точка останова должна быть достигнута только N-раз, когда достигнуто условие или каждые N-

Фильтрация по Machine/Thread/Process (Имя машины/Поток/Процесс)

Можно также щелкнуть правой кнопкой на точке останова и выбрать пункт «Filter...» («Фильтр...») из контекстного меню, чтобы указать, что точка останова должна быть достигнута, если процесс отладки происходит на определенной машине или в определенном процессе или в определенной потоке.

Точки трассировки (TracePoints) – пользовательские события при попадании в точку останова

Одной из функций отладчика является возможность использовать **TracePoints (Точки трассировки)**.

Точка трассировки это точка останова при достижении которой срабатывает пользовательское событие. Эта функциональность особенно полезна, когда вам нужно отследить поведение в вашем приложении без остановки в отладке.

Установка точки трассировки (TracePoint)

Можно включить точки трассировки воспользовавшись клавишей F9 для установки точки останова в строке кода и затем щелкнув правой кнопкой на точке останова и выбрав пункт «When Hit...» («Когда Попадает...») из контекстного меню.

Это приведет к появлению диалогового окна, которое позволяет вам указать, что должно произойти когда достигнута точка останова

Пошаговое выполнение программы

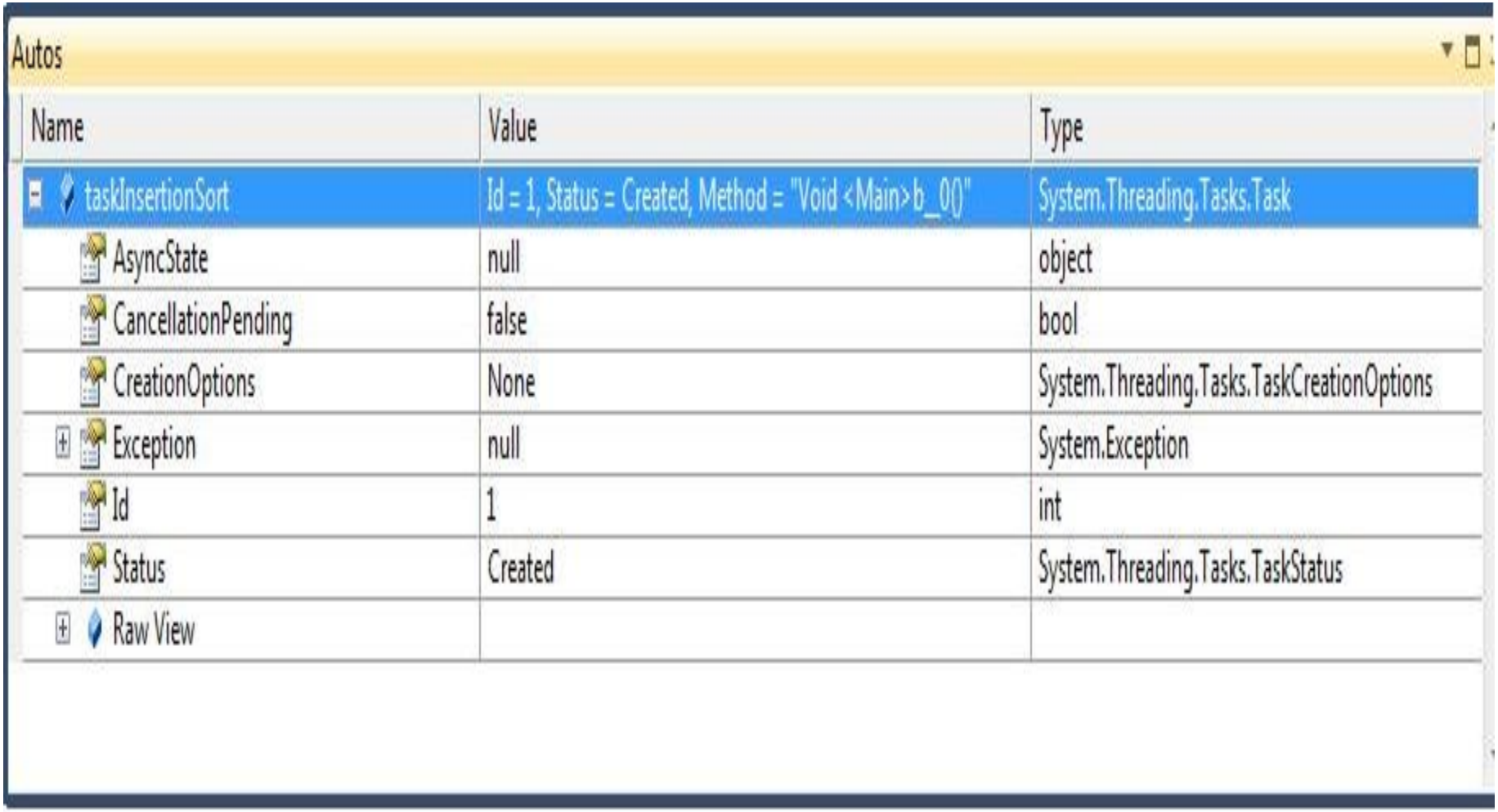
При выполнении пошаговой отладки, разработчик может использовать следующие окна для просмотра значение переменных или если приложение многопоточное, то просматривать состояние потоков или переключаться между ними:

- Autos;
- Locals;
- Watch;
- Immediate;
- Threads;
- Parallel Task;
- Parallel Stacks.

Окно Autos

Окно Autos используется для того чтобы, просматривать значения, связанные с той строкой кода, на которой находится курсор отладки. Это окно отображает значения всех переменных и выражений, имеющихся в текущей выполняющейся строке кода или в предыдущей строке кода. Содержит следующие столбцы:

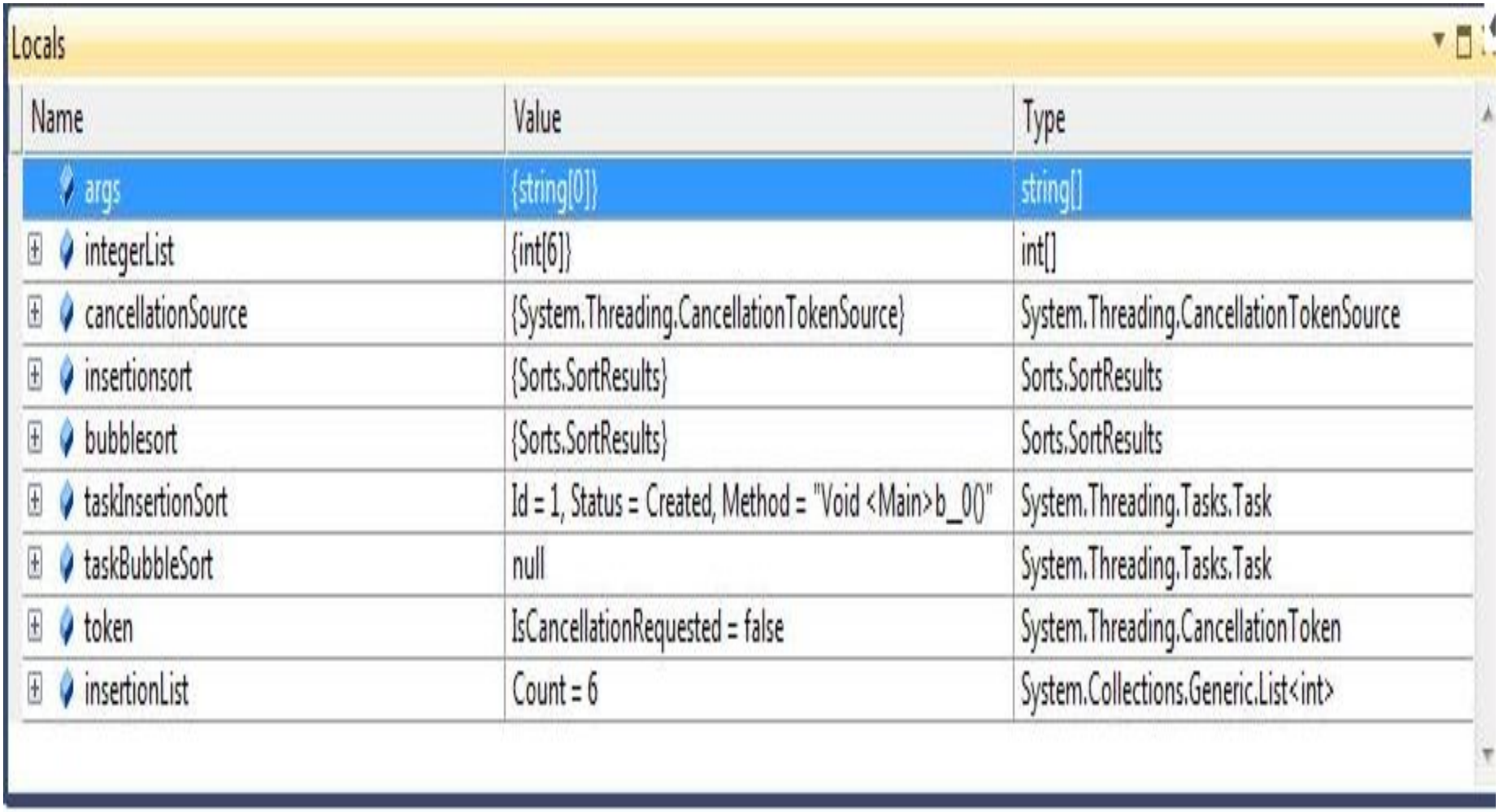
- Name - название переменной;
- Value - значение переменной;
- Type - тип переменной.


Autos
Name
Value
Type
taskInsertionSort
Id = 1, Status = Created, Method = "Void <Main>b_0()"
System.Threading.Tasks.Task
AsyncState
null
object
CancellationPending
false
bool
CreationOptions
None
System.Threading.Tasks.TaskCreationOptions
Exception
null
System.Exception
Id
1
int
Status
Created
System.Threading.Tasks.TaskStatus
Raw View

Окно Locals

Окно Locals отображает все переменные и их значения для текущей области видимости отладчика, что дает представление обо всех переменных, которые используются в текущем выполняющемся методе. Переменные в этом окне автоматически настраиваются отладчиком. Данное окно содержит следующие столбцы:

- Name - название переменной;
- Value - значение переменной;
- Type - тип переменной.


Locals
Name
Value
Type
args
{string[0]}
string[]
integerList
{int[6]}
int[]
cancellationSource
{System.Threading.CancellationTokenSource}
System.Threading.CancellationTokenSource
insertionsort
{Sorts.SortResults}
Sorts.SortResults
bubblesort
{Sorts.SortResults}
Sorts.SortResults
taskInsertionSort
Id = 1, Status = Created, Method = "Void <Main>b_0()"
System.Threading.Tasks.Task
taskBubbleSort
null
System.Threading.Tasks.Task
token
IsCancellationRequested = false
System.Threading.CancellationToken
insertionList
Count = 6
System.Collections.Generic.List<int>

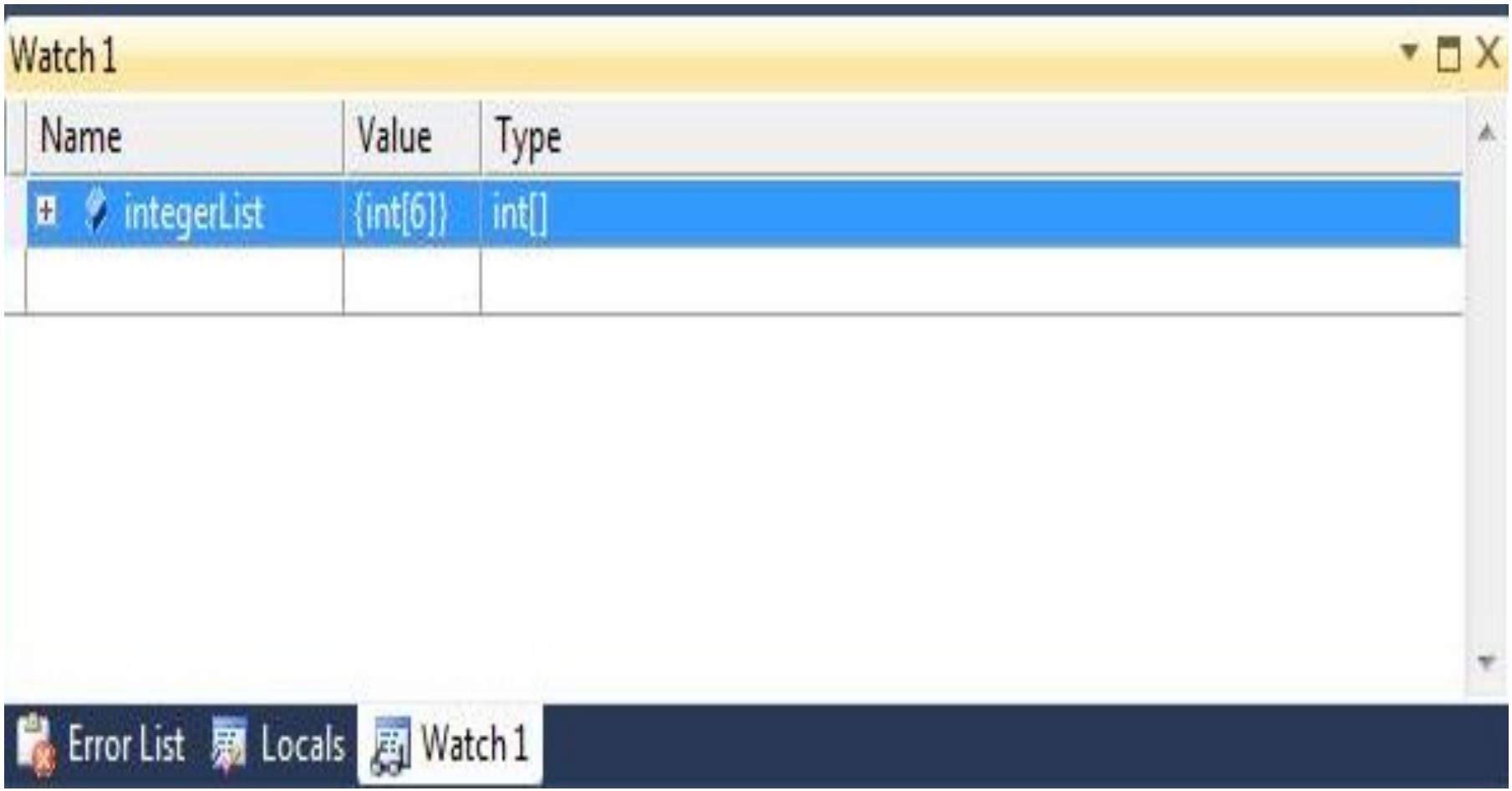
Окно Watch

Окно Watch или окно контрольных значений - позволяет настраивать собственный список переменных и выражений, которые необходимо отслеживать. Всего доступно четыре окна Watch (Watch 1, Watch 2, Watch 3 и Watch 4), что позволяет выделить в четыре списка переменные и выражения, данную возможность удобно использовать в том случае, если каждый список относится к отдельной области видимости приложения.

Переменные или выражение в окно Watch добавляются или из редактора кода, или из окна QuickWatch. Если нужно добавить в окно Watch элемент из редактора кода, то нужно выделить нужную переменную или выражение, щелкнуть по ней правой кнопкой мыши и выбирать пункт Add Watch.

Также можно перетаскивать, с помощью мыши, выделенный элемент в окно Watch. Данное окно содержит следующие столбцы:

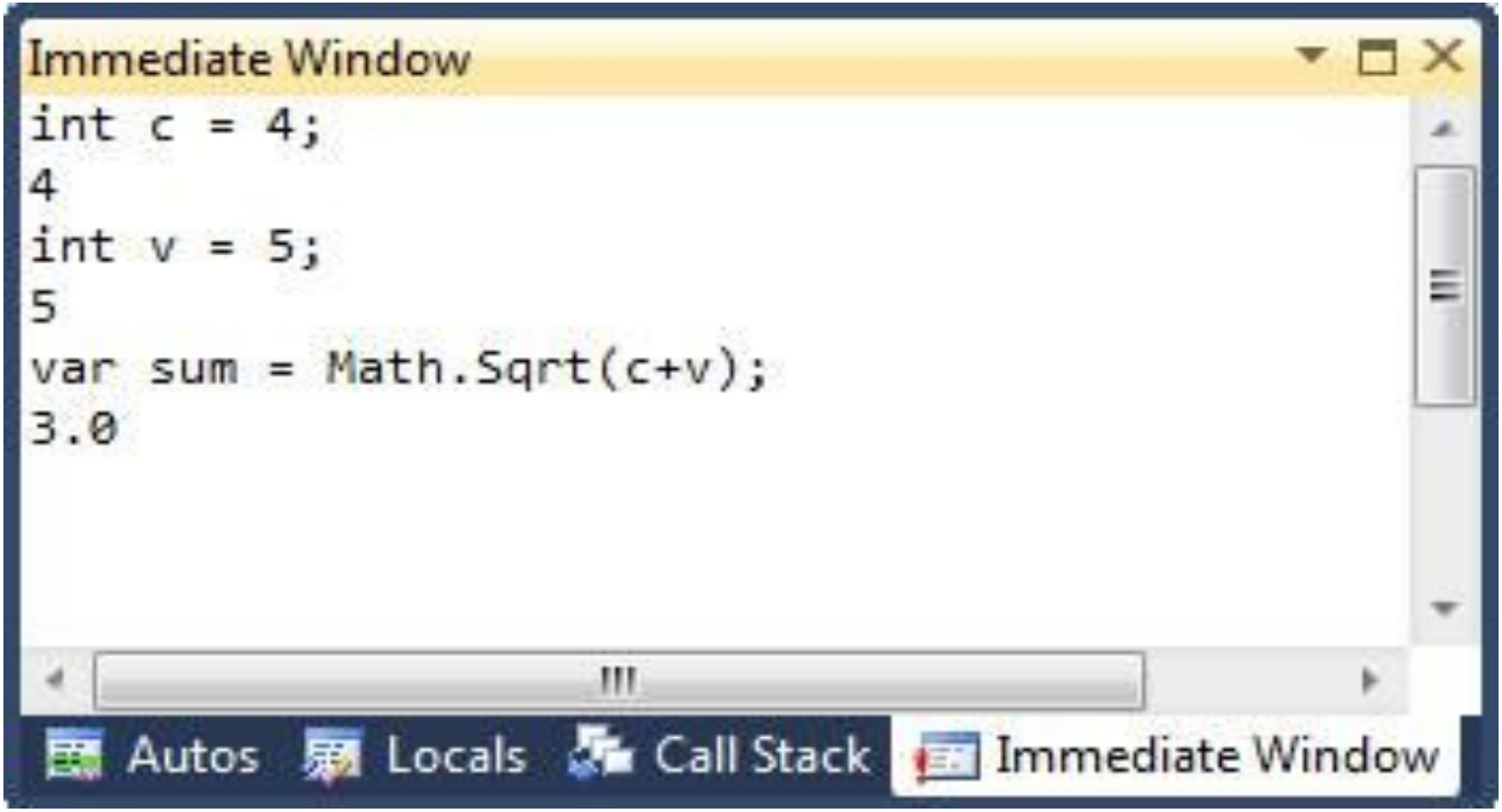
- Name - название переменной;
- Value - значение переменной;
- Type - тип переменной.


Watch 1
Name
Value
Type
integerList
{int[6]}
int[]
Error List
Locals
Watch 1

Окно Immediate

Окно Immediate или непосредственное выполнение - предназначено для ручного ввода и выполнения команд. Это окно появляется автоматически при прерывании работы программы в точках останова программы.

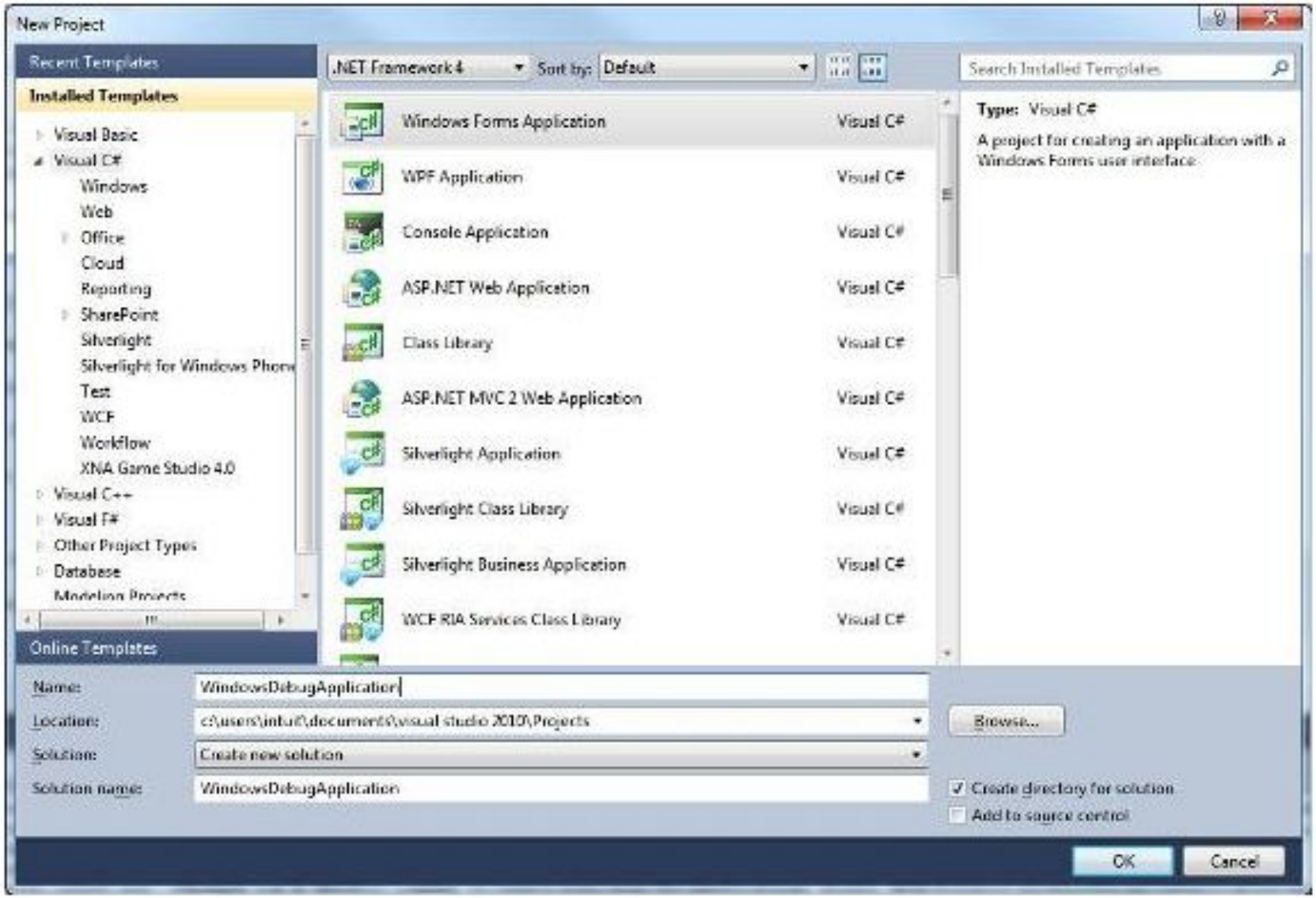
Для выполнения команды или оператора необходимо написать команду и нажать клавишу <Enter>.


Immediate Window
int c = 4;
4
int v = 5;
5
var sum = Math.Sqrt(c+v);
3.0
Autos
Locals
Call Stack
Immediate Window

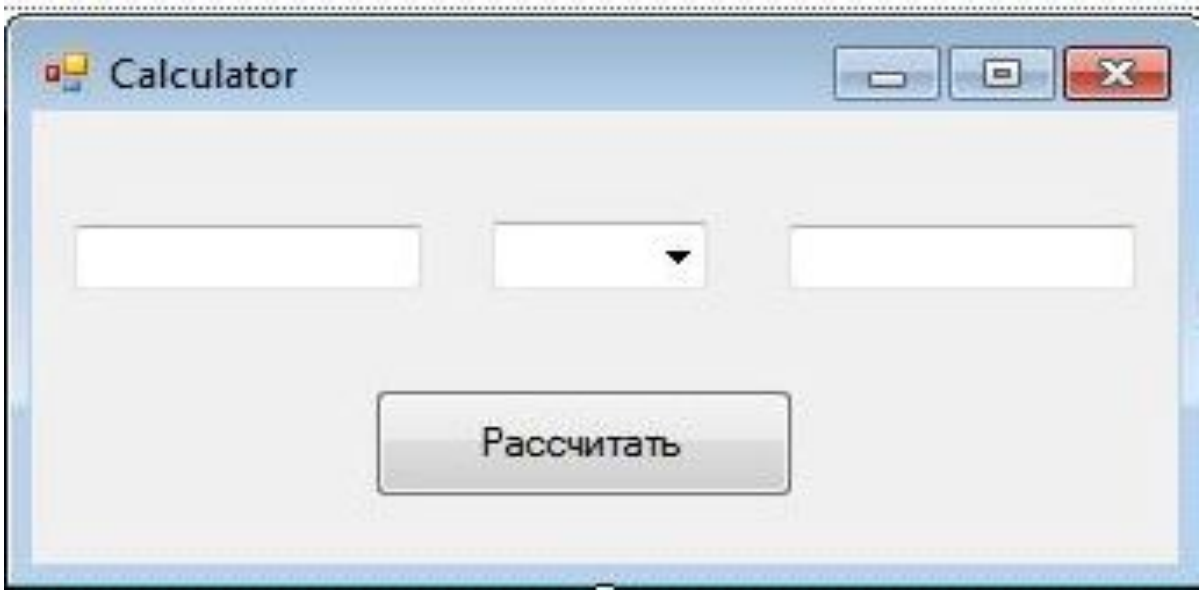
Отладка однопоточного приложения в Visual Studio 2010

Рассмотрим процесс создания и отладки однопоточного приложения с использованием Visual Studio 2010.

1. Создадим Windows приложение с названием "WindowsDebugApplication"


New Project
Recent Templates
Installed Templates
Visual Basic
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Silverlight for Windows Phone
Test
WCF
Workflow
XNA Game Studio 4.0
Visual C++
Visual F#
Other Project Types
Database
Modeling Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Application Visual C#
Class Library Visual C#
ASP.NET MVC 2 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Silverlight Business Application Visual C#
WCF RIA Services Class Library Visual C#
Type: Visual C#
A project for creating an application with a Windows Forms user interface
Name: WindowsDebugApplication
Location: c:\users\intuit\documents\visual studio 2010\Projects
Browse...
Solution: Create new solution
Solution name: WindowsDebugApplication
Create directory for solution
Add to source control
OK
Cancel

2. Создадим простейший калькулятор. Для этого разместим на форме 4 элемента (2 TextBox, 1 ComboBox, 1 Button):



3. Добавим в программу код.

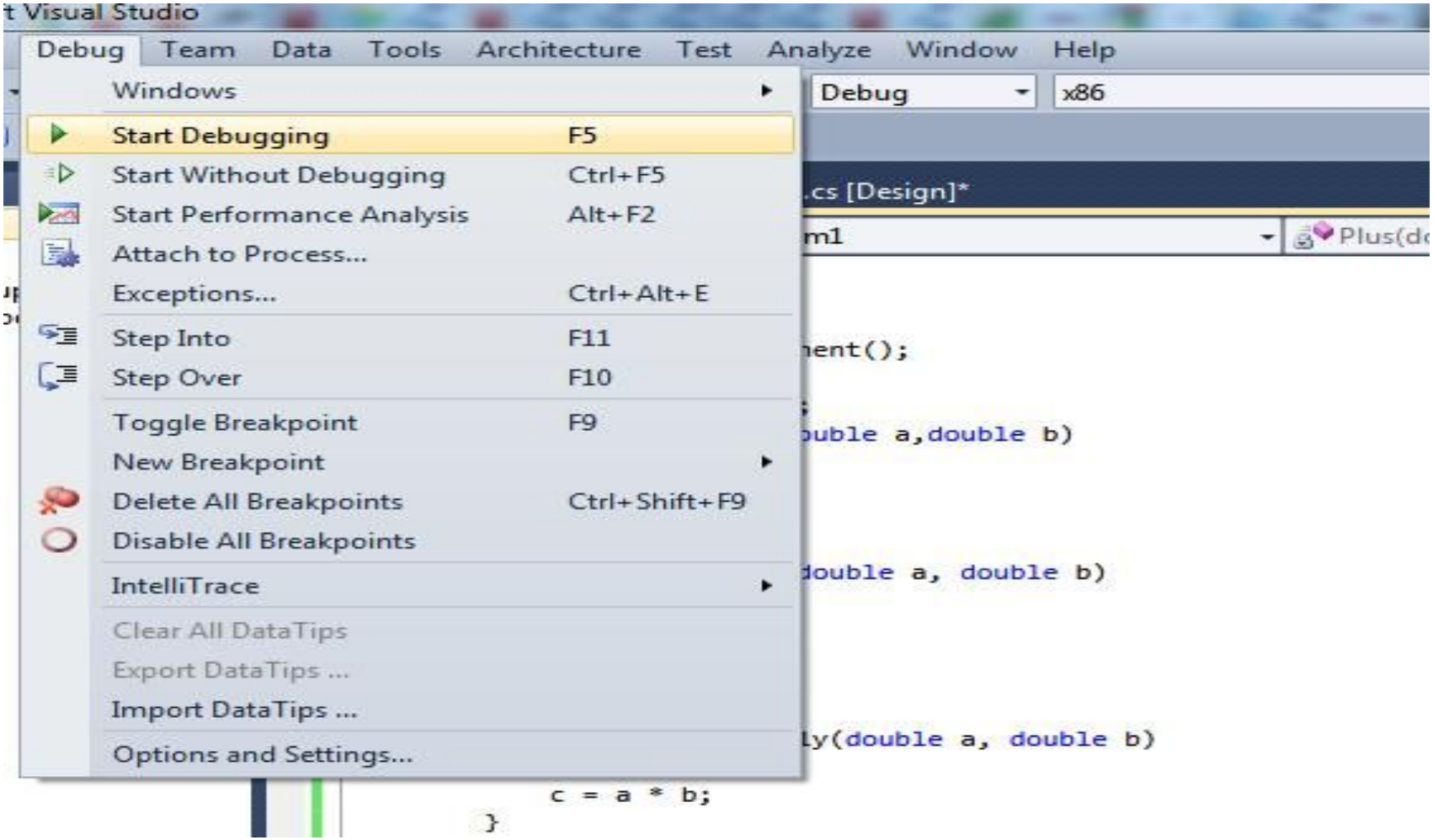
4. Теперь, расставим точки останова (breakpoints) в программе. В примере точки останова расставлены напротив методов математических операций и в событии кнопки:

Program.cs | Form1.cs* | Form1.cs [Design]*

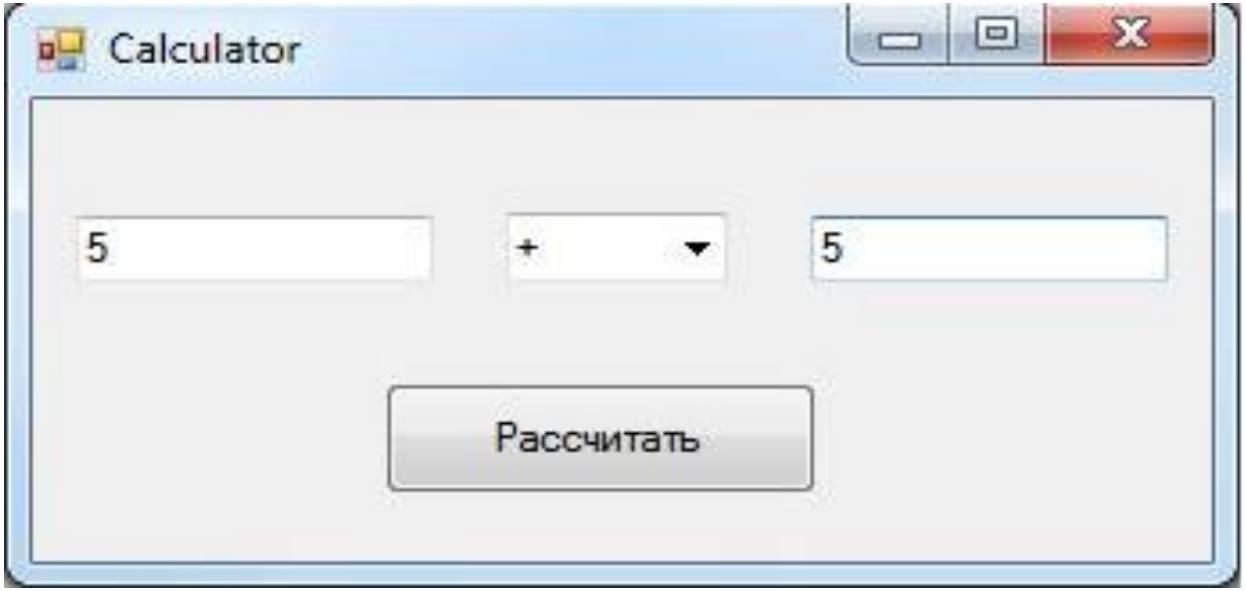
WindowsDebuggingApplication.Form1 | Plus(double a, dout

```
    {
        public Form1()
        {
            InitializeComponent();
        }
        private double c =0;
        private void Plus(double a,double b)
        {
            c = a + b;
        }

        private void Minus(double a, double b)
        {
            c = a - b;

        }

        private void Multiply(double a, double b)
        {
            c = a * b;
        }

        private void Split(double a, double b)
        {
            c = a / b;
        }

        private void ResultButton_Click(object sender, EventArgs e)
        {
            string symbol =(string) SymbolcomboBox.SelectedItem;
            ArrayList arraysymbol = new ArrayList();
            foreach (var s in SymbolcomboBox.Items)
            {
                arraysymbol.Add(s);
            }

            double var1 =Convert.ToDouble(FirstVariable.Text);
            double var2 = Convert.ToDouble(SecondVariable.Text);
            switch (symbol)
```

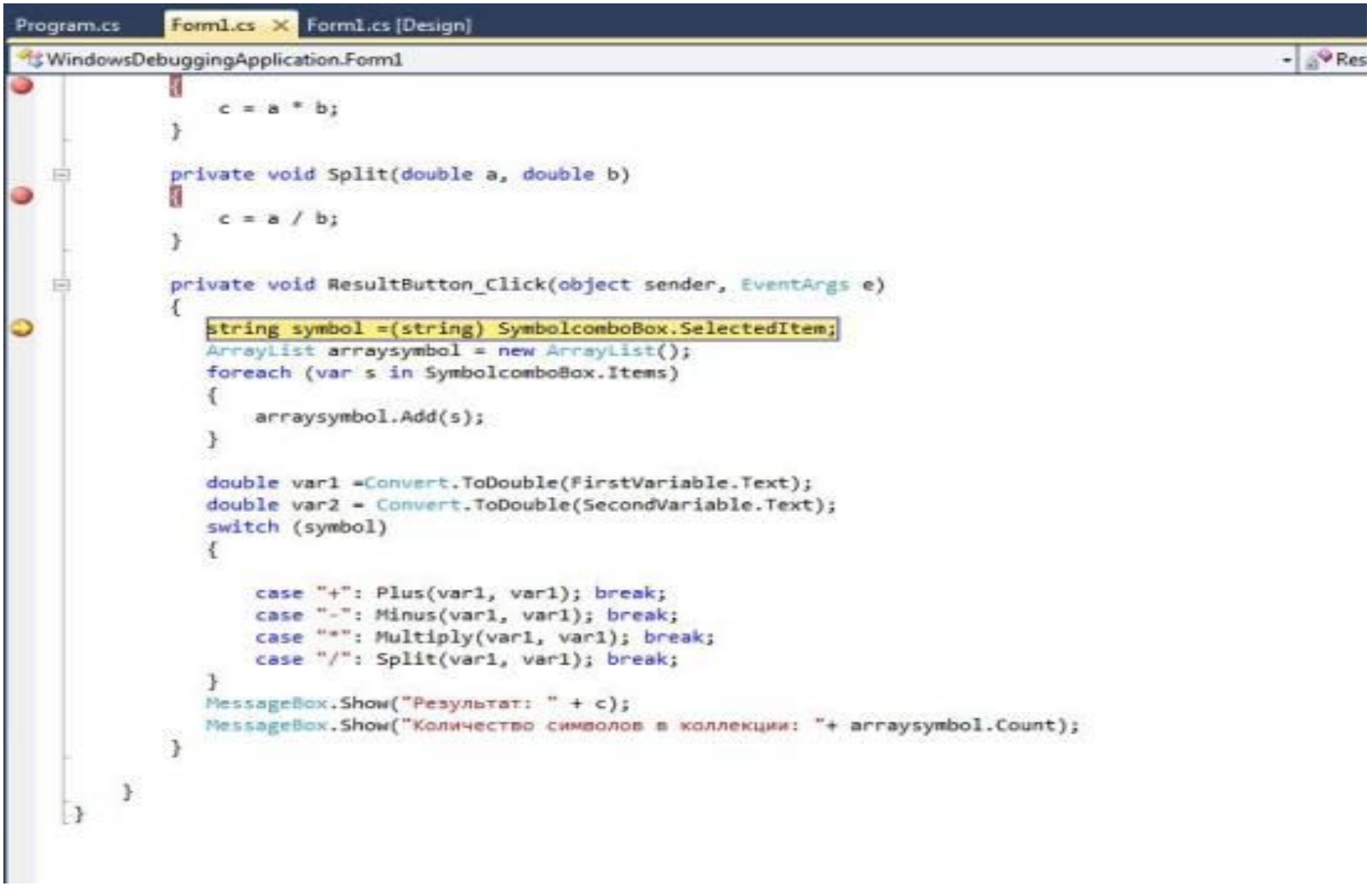
5. Запускаем отладку приложения с помощью пункта "Start Debugging" - меню "Debug" или с помощью клавиши "F5":



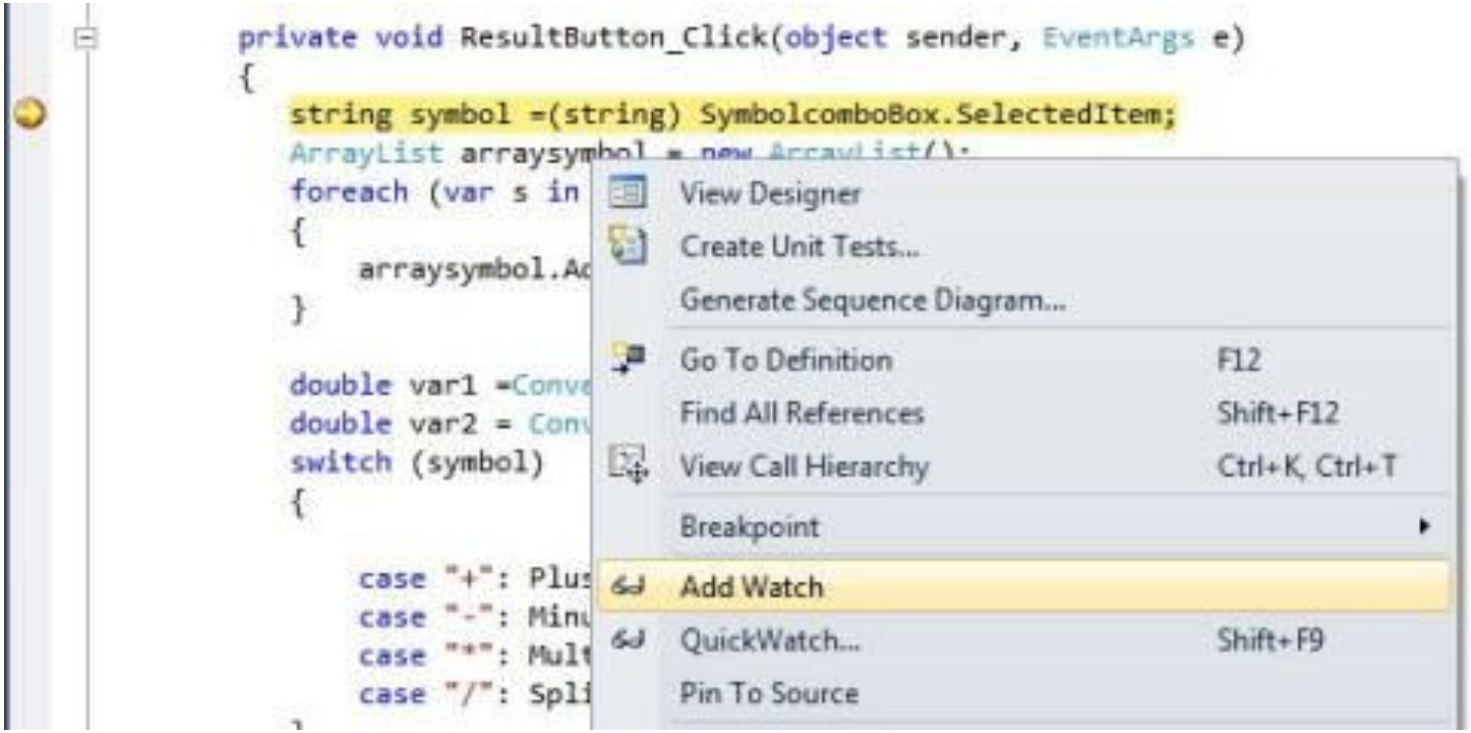
6. После ввода значений в поля программы и выбора соответствующей операции (сложение, вычитание и т.д), жмем кнопку "Рассчитать", тем самым вызовется метод ResulButton_Click():



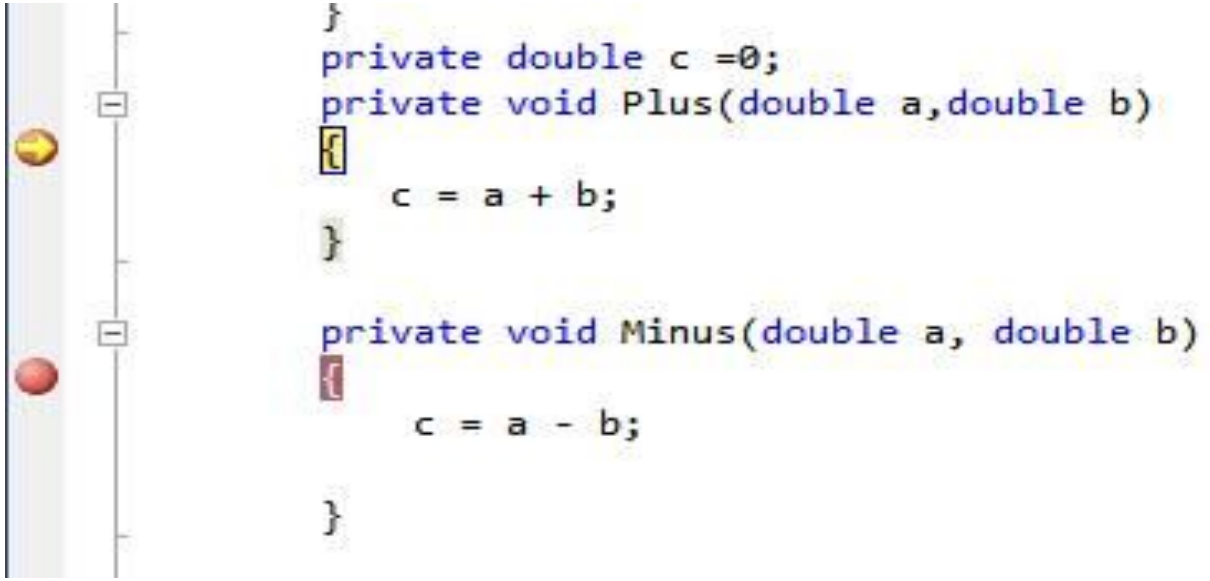
Запустится пошаговый процесс отладки приложения с точки останова (Breakpoint) в методе ResulButton_Click():



7. Добавим для просмотра значений переменных - переменные arraysymbol (коллекция), и переменную "c". Для этого щелкните на нужной переменной правой кнопкой мыши и выберите из списка "Add Watch":

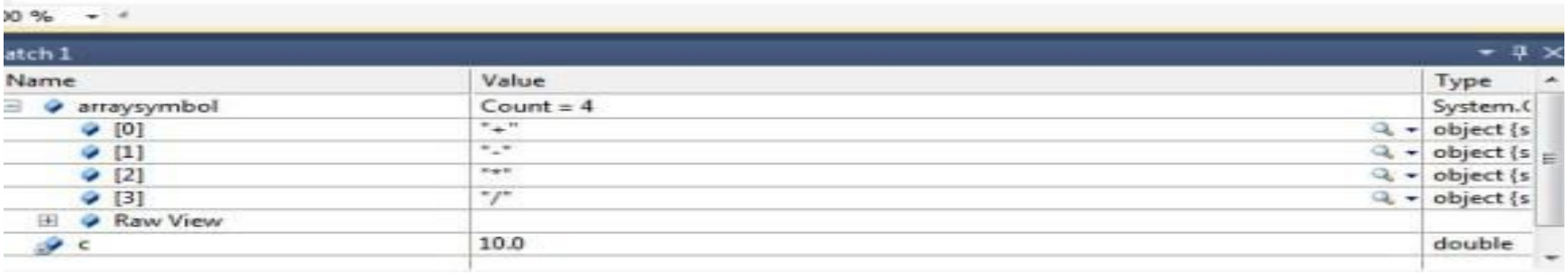


8. Используйте кнопку "F10" для пошаговой отладки приложения. В процессе пошаговой отладки, курсор отладчика будет заходить в те методы, которые вызываются в методе ResulButton_Click(), в нашем случае это метод Plus() (т.к была выбрана операция сложения - "+"):

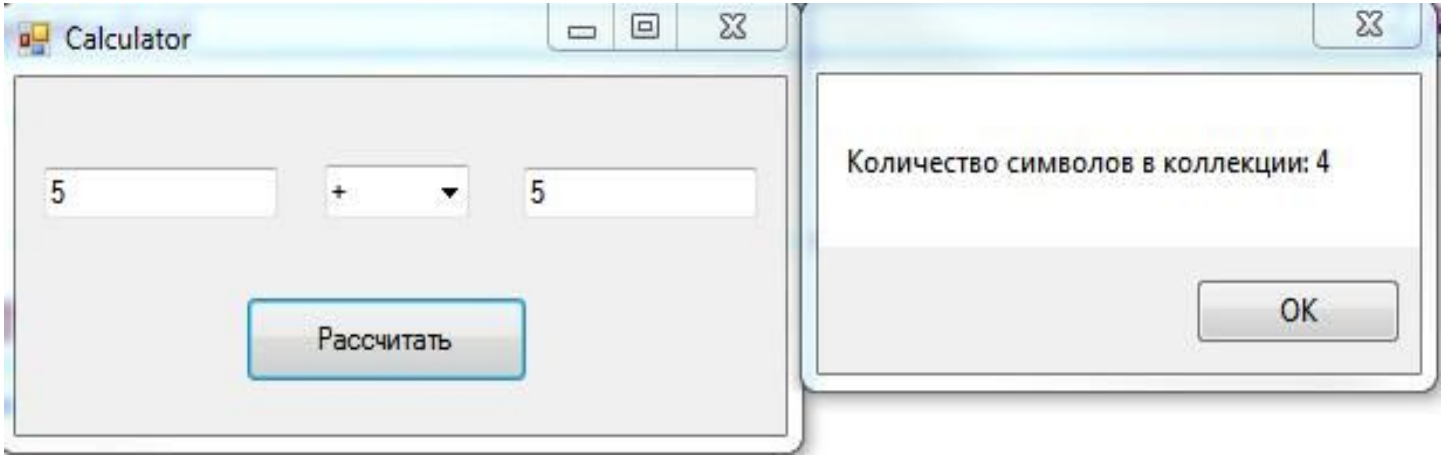


В процессе отладки приложения, значения переменных, в списке Watch, будут изменяться в зависимости от шага:

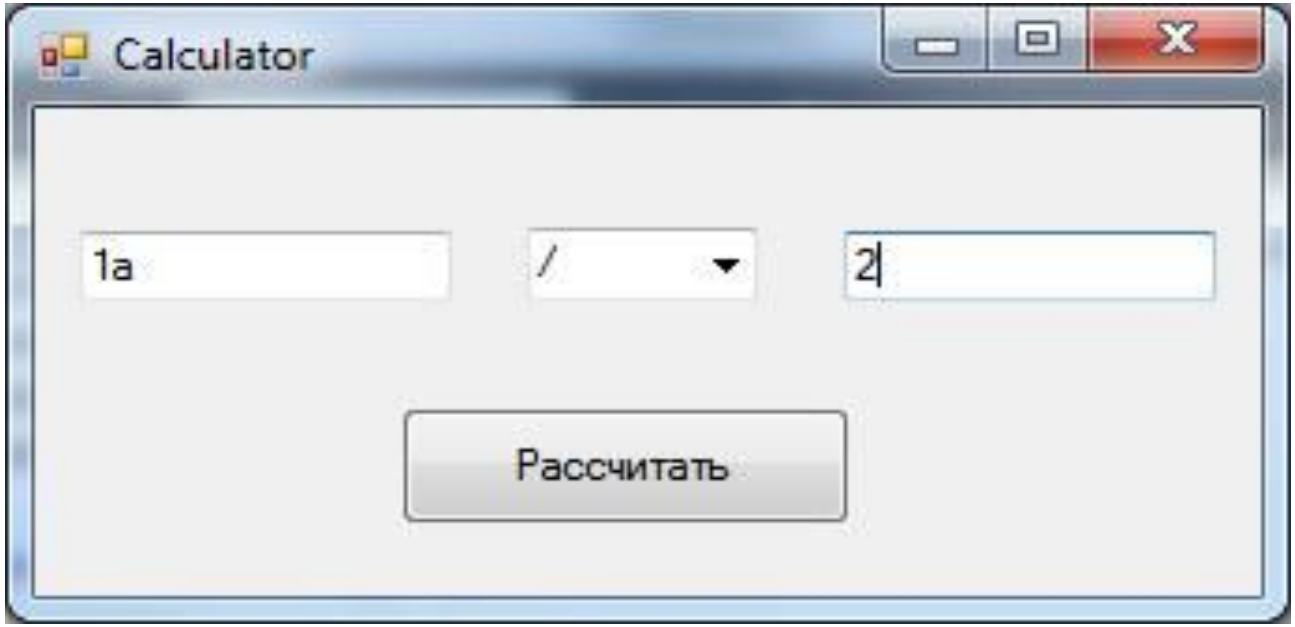
```
        private void ResultButton_Click(object sender, EventArgs e)
        {
            string symbol =(string) SymbolcomboBox.SelectedItem;
            ArrayList arraysymbol = new ArrayList();
            foreach (var s in SymbolcomboBox.Items)
            {
                arraysymbol.Add(s);
            }

            double var1 =Convert.ToDouble(FirstVariable.Text);
            double var2 = Convert.ToDouble(SecondVariable.Text);
            switch (symbol)
            {

                case "+": Plus(var1, var1); break;
                case "-": Minus(var1, var1); break;
                case "*": Multiply(var1, var1); break;
                case "/": Split(var1, var1); break;
            }
            MessageBox.Show("Результат: " + c);
            MessageBox.Show("Количество символов в коллекции: "+ arraysymbol.Count);
        }

    }
}
```



10. После завершения отладки (и если не возникло не каких ошибок) программа выдаст результаты:

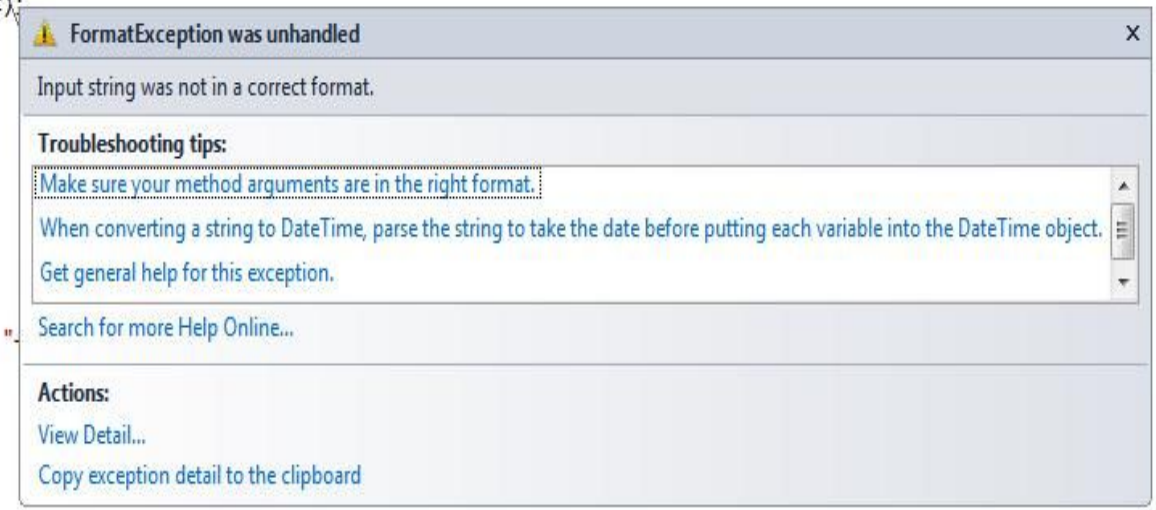


11. Повторно запустим отладку приложения и намеренно введем значения, вызывающие исключение. В нашем случае это "1a" и "2":



Если в программе не обрабатываются исключения, отладчик выдаст ошибку, на строке, где возникает исключение. В нашем случае исключение, связанное с преобразованием формата типа string в формат double:

```
        private void ResultButton_Click(object sender, EventArgs e)
        {
            string symbol =(string) SymbolcomboBox.SelectedItem;
            ArrayList arraysymbol = new ArrayList();
            foreach (var s in SymbolcomboBox.Items)
            {
                arraysymbol.Add(s);
            }

            double var1 =Convert.ToDouble(FirstVariable.Text);
            double var2 = Convert.ToDouble(SecondVariable.Text);
            switch (symbol)
            {

                case "+": Plus(var1, var1); break;
                case "-": Minus(var1, var1); break;
                case "*": Multiply(var1, var1); break;
                case "/": Split(var1, var1); break;
            }
            MessageBox.Show("Результат: " + c);
            MessageBox.Show("Количество символов в коллекции: "
        }

    }
}
```



12. Для того что бы остановить отладку используйте пункт меню "Stop Debugging" меню Debug или используя сочетание клавиш "Shift+F5«.