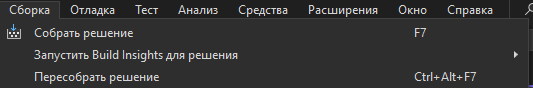
Выполнил студент группы ВМК-22 Александров Данил

Конспект и отчёт по Visual Studio

**Режимы запуска и компиляции: Сборка, запуск без отладки, запуск с отладкой.**

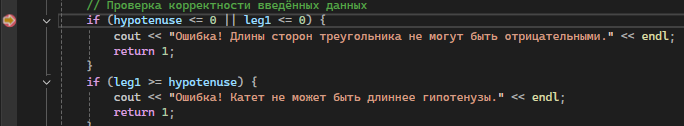
Запуск без отладки  
  
Это процесс запуска программы без привязки к отладчику. Это значит, что вы не сможете остановить выполнение программы на определенном месте, пошагово просматривать код и проверять значения переменных в режиме реального времени. Запуск без отладки полезен, когда вам не требуется проводить подробное ручное тестирование вашего приложения или когда вы хотите проверить его поведение без прерываний.

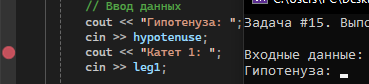
Запуск с отладкой  
  
В контексте Visual Studio отладка приложения обычно означает запуск приложения с подключенным отладчиком (то есть в режиме отладчика). При этом в отладчике доступно множество способов наблюдения за выполнением кода. Вы можете выполнить шаги по коду и просмотреть значения, хранящиеся в переменных, вы можете настроить контроль над переменными, чтобы увидеть, когда изменяются значения, можно проверить путь выполнения кода и т. д.

Сборка  
   
Сборка представляет собой процесс компиляции исходного кода в исполняемый файл или библиотеку, которые могут быть запущены на компьютере пользователя.

Во время сборки решения, Visual Studio компилирует исходные файлы проекта, проверяет на наличие ошибок, объединяет все необходимые зависимости и генерирует исполняемые файлы. Для запуска нужно выбрать в меню "Сборка" и нажать на пункт "Собрать решение", наш проект скомпилируется и в папке проекта появится наш тестовый исполняемый файл ("exe").

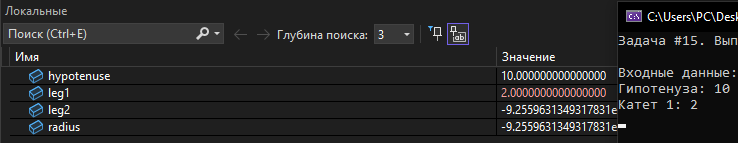
**Отладка в VS. Точки останова. Режимы выполнения. Просмотр значений переменных. Стек вызовов.**

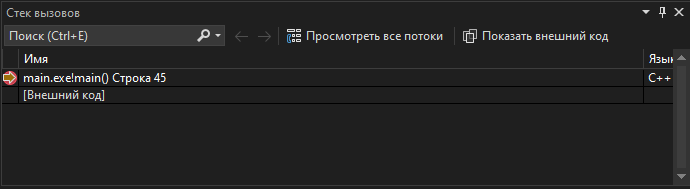
Точки останова  
Точки останова полезны, если вам известны строка или раздел кода, которые вы хотите подробно изучить в среде выполнения. Точки останова — это самая основная и важная функция надежной отладки. Точка останова указывает, где Visual Studio следует приостановить выполнение кода, чтобы вы могли проверить значения переменных или поведение памяти либо выполнение ветви кода. Если вы открыли файл в редакторе кода, точку останова можно задать, щелкнув в поле слева от строки кода.

Режимы выполнения  
Нажмите клавишу F5 (Отладка > запуска отладки) или кнопку "Начать отладку" на панели инструментов отладки, и отладчик запускается в первую точку останова, с которой он сталкивается.

  
Для запуска приложения с подключенным отладчиком нажмите клавишу F11 (Отладка > Шаг с заходом). F11 — это команда Шаг с заходом, которая выполняет приложение с переходом к следующему оператору. При запуске приложения с помощью клавиши F11 отладчик останавливается на первом выполняемом операторе.  
 Желтая стрелка представляет оператор, на котором приостановлен отладчик. В этой же точке приостанавливается выполнение приложения (этот оператор пока не выполнен).  
Когда вы находитесь в строке кода, представляющей собой вызов функции или метода, можно нажать клавишу F10 (Отладка > Шаг с обходом) вместо F11. Клавиша F10 продолжает выполнение отладчика без захода в функции или методы в коде приложения (код продолжает выполняться).

  
Использование кнопки Выполнение до щелкнутого аналогично установке временной точки останова. Кроме того, эта команда удобна для быстрой работы в видимой области кода приложения.

Просмотр значений переменных  
В окне Локальные отображаются переменные вместе с текущим значением и типом. Окно Видимые показывает все переменные, используемые в текущей или предыдущей строке

Стек вызовов  
В окне Стек вызовов показан порядок вызова методов и функций. В верхней строке показана текущая функция. Во второй строке показана функция или свойство, из которого она вызывалась, и т. д. Стек вызовов хорошо подходит для изучения и анализа потока выполнения приложения

**Файлы и папки проекта VisualStudio.  
Какие файлы и папки стоит добавлять к отслеживанию гитом?**

red1:

Файлы и папки проекта VisualStudio.

|  |  |  |
| --- | --- | --- |
| Расширение | Имя | Описание |
| .sln | Файл решения Visual Studio | Организует проекты, элементы проектов и решений в решении. |
| .suo | Файл параметров пользователя решения | Сохраняет настройки и параметры уровня пользователя, например точки останова. |
| Папка | x64 | Внутри этой папки хранятся файлы, относящиеся к компиляции и сборке проекта для 64-битных систем. Здесь могут находиться скомпилированные исполняемые файлы, объектные файлы, библиотеки и другие сборочные артефакты, относящиеся к 64-битной архитектуре. |
| .user | Файл пользовательских настроек | Файлы с расширением .user содержат пользовательские настройки проекта, которые могут быть уникальными для каждого разработчика. Обычно эти файлы могут содержать настройки отладчика, окон, расположения панелей и так далее. Они позволяют установить персональные настройки, не влияющие на общий проект. |
| .vcproj | Файлы проекта. | Файлы с расширением .vcproj представляют собой проектные файлы C++ в формате, используемом в более ранних версиях Visual Studio (например, в Visual Studio 2005, 2008 и т. д.). Они содержат информацию о конфигурации проекта, файлы, которые включены в проект, зависимости, параметры компилятора и другую настройку проекта. |
| .vcxproj | Файл проекта. | Хранит информацию, относящуюся к каждому проекту. (В более ранних версиях этот файл был назван Projname.vcproj или Projname.dsp.) Пример файла проекта C++ (.vcxproj) см. в разделе "Файлы проекта". |
| .vcxitems | Файл проекта общих элементов. | Этот проект не создается. Вместо этого на него может сослаться другой проект C++, и его файлы станут частью процесса сборки ссылающегося проекта. Это можно использовать для совместного использования общего кода в кроссплатформенных проектах C++. |
| .sdf | Файл базы данных просмотра. | Поддерживает возможности просмотра и навигации, такие как Перейти к определению, Найти все ссылки и Представление классов. Создается путем анализа файлов заголовков. |
| .vcxproj.filters | Файл фильтров. | Указывает, куда поместить файл, который добавляется в решение. Например, H-файл помещается в узел Файлы заголовков. |
| .vcxproj.user | Файл пользователя миграции. | После миграции проекта из Visual Studio 2008 в этом файле появляются данные, преобразованные из любых VSPROPS-файлов. |
| .idl | Файл для конкретных проектов | Содержит исходный код на языке описания интерфейсов (IDL) для библиотеки типов элементов управления. Используется Visual C++ для создания библиотеки типов. Созданная библиотека предоставляет доступ к интерфейсу элемента управления другим клиентам автоматизации. Дополнительные сведения см. в разделе Файл определения интерфейса (IDL-файл) для пакета Windows SDK. |
| Readme.txt | Файл сведений | Создается мастером приложений и описывает файлы в проекте. |

**Какие файлы и папки стоит добавлять к отслеживанию гитом?**

Основные принципы добавления файлов и папок в Git

1. Исходныйкодпроекта — это главный компонент, который обязательно должен быть отслеживаемым. Git используется для отслеживания изменений в исходных файлах программы, таких как .cpp, .h, .cs, .js и другие файлы, содержащие код.
2. Конфигурационныефайлы — это файлы, определяющие настройки проекта, среды разработки и сборки. К таким файлам относятся:

* CMakeLists.txt (для проектов на C++)
* .csproj, .sln (для проектов на C#)
* .json, .xml, .yaml для конфигураций Эти файлы важны для правильной настройки среды и сборки проекта, и их также стоит отслеживать.

1. Ресурсыпроекта — сюда относятся изображения, шрифты, аудиофайлы и другие медиафайлы, используемые в проекте. Например:

* .png, .jpg, .svg — графика
* .mp3, .ogg — звуки Эти файлы необходимы для правильного функционирования приложения, и их нужно добавлять в репозиторий.

1. Тестыидокументация — все файлы, которые связаны с тестированием проекта и его документацией, должны отслеживаться. Это могут быть:

* Исходные файлы тестов
* Документация в формате .md (например, README.md), .txt, .rst и другие.

1. Скрипты — если в проекте используются скрипты для автоматизации, например, для установки зависимостей или сборки проекта, их также следует отслеживать в репозитории:

* Скрипты на Bash, PowerShell, Python и другие.

**Что не стоит добавлять в репозиторий**

1. Файлы**,** генерируемыесборкой — это файлы, которые создаются в процессе сборки проекта, и которые можно восстановить в любой момент. К таким файлам относятся:

* Бинарные файлы (.exe, .dll, .so, .obj, .a)
* Лог-файлы сборки
* Каталоги, содержащие скомпилированные файлы (например, bin/, obj/, build/)

1. Файлынастроексредыразработки — это файлы, которые могут быть специфичными для локальной среды разработчика и не должны быть общими для всего проекта. Например:

* Файлы настроек IDE (например, .vscode/, .idea/, .vs/)
* Локальные файлы конфигурации (.user, .userosscache)

1. Файлы**,** содержащиечувствительныеданные — файлы, которые содержат пароли, ключи API, конфиденциальную информацию или другие чувствительные данные, не должны попадать в репозиторий. Эти файлы следует добавлять в .gitignore или использовать для них специальные решения (например, Git-скрипты для управления секретами).
2. Кэшированныефайлы — временные файлы и папки, которые создаются при работе с проектом, например, кэшированные изображения или другие временные данные. Эти файлы могут быть указаны в .gitignore для исключения из репозитория.

**Рефакторинг в VS**

Рефакторинг — это процесс улучшения структуры исходного кода программы без изменения её функциональности. Основная цель — улучшить читаемость, поддерживаемость и производительность кода, а также уменьшить количество багов.

**Зачем нужен рефакторинг?**

* Читаемость: Упрощение кода делает его более понятным для других разработчиков и для вас в будущем.
* Поддерживаемость: Легче вносить изменения в чистый и структурированный код.
* Устранениедублирования: Меньше повторяющихся участков кода, что снижает шанс на ошибку.
* Оптимизация: Улучшение производительности и удаление избыточного кода.

**Виды рефакторинга**

* Мелкиеизменения: Переименование переменных и методов, форматирование кода.
* Оптимизация: Устранение избыточного кода, улучшение структуры данных, замена устаревших или неэффективных методов.
* Реорганизация: Перемещение кода в более подходящие места (например, выделение функций или классов).

**Инструменты рефакторинга в Visual Studio**

Visual Studio предоставляет множество инструментов для автоматизации и облегчения рефакторинга:

a) Переименование (Rename)

* Для переименования переменной, метода, класса и т. д., используйте функционал переименования с поддержкой обновления всех ссылок на элемент.
  + Правый клик на элементе → Rename или Ctrl + R, R.

b) Изменение сигнатуры метода (Change Signature)

* Удобный инструмент для изменения сигнатуры метода, добавления или удаления параметров.
  + Правый клик на методе → Change Signature.

c) Извлечение метода (Extract Method)

* Выделяет фрагмент кода в новый метод, улучшая читаемость.
  + Выделите участок кода → Refactor → ExtractMethod.

d) Извлечение класса (Extract Class)

* Перемещает часть логики в новый класс для лучшей организации.
  + Правый клик → Refactor → ExtractClass.

e) Интерфейсы и абстракции (Implement Interface, Implement Abstract Class)

* Visual Studio помогает реализовать интерфейсы и абстрактные классы с помощью автоматических шаблонов кода.
  + Правый клик на классе → ImplementInterface или ImplementAbstractClass.

f) Удаление неиспользуемых переменных (Remove Unused Code)

* Инструменты для поиска и удаления неиспользуемых переменных, методов или классов.
  + FindUnusedCode в меню Refactor.

**Стратегии рефакторинга**

* Маленькимишагами: Изменяйте код поэтапно, чтобы не нарушить его логику.
* Тестирование: После каждого изменения важно запускать тесты, чтобы убедиться, что рефакторинг не нарушил функциональность программы.
* Использованиеверсионныхсистем: Например, Git, чтобы можно было легко откатить изменения в случае ошибки.

**Преимущества рефакторинга в Visual Studio**

* ИнтеграциясIDE: Весь процесс рефакторинга осуществляется в среде разработки, что упрощает процесс и минимизирует ошибки.
* Автоматизация: Множество задач можно выполнить автоматически, экономя время.
* Поддержкатестов: В Visual Studio интегрированы инструменты для работы с юнит-тестами, что помогает быстро проверять изменения.

**todo комментарии. Зачем нужны? Как создать? Как посмотреть список?**

**Зачем нужны TODO комментарии?**

TODO комментарии используются в программировании для отметки мест в коде, где нужно выполнить дополнительную работу или улучшения в будущем. Это позволяет разработчикам оставить напоминания для себя или других участников команды о том, что какой-то участок кода требует доработки, улучшения или исправления. TODO комментарии могут быть полезными в следующих случаях:

* Для временных решений, которые должны быть улучшены позже.
* Для указания участков кода, которые требуют доработки или оптимизации.
* Для обозначения частей кода, которые должны быть реализованы после выполнения других задач.
* В проектах с несколькими разработчиками для уведомления о необходимых изменениях.

**Как создать TODO комментарий?**

Создать TODO комментарий очень просто — достаточно добавить комментарий в код, содержащий ключевое слово "TODO". В большинстве языков программирования это делается следующим образом:

* C++/C#: **//** TODO: Добавить обработку ошибок
* Python: # TODO: Написать тесты для этой функции
* JavaScript: // TODO: Реализовать алгоритм сортировки
* HTML: <!-- TODO: Добавить валидацию формы -->

**Как посмотреть список TODO комментариев?**

Многие интегрированные среды разработки (IDE) и редакторы поддерживают возможность поиска и отображения всех TODO комментариев в проекте:

* VisualStudio**:** В Visual Studio можно воспользоваться встроенным инструментом "Task List" для поиска TODO комментариев. Для этого нужно открыть "View" -> "Task List", и IDE автоматически покажет все TODO и другие помеченные задачи.
* VS Code**:** В VS Code можно использовать команду поиска (Ctrl + Shift + F) и искать по строке "TODO". Это отобразит все комментарии с TODO в проекте.

**Как управлять TODO комментариями?**

* TODO комментарии должны быть удалены или выполнены после того, как задача будет завершена. Это позволяет поддерживать код в чистоте.
* Используйте TODO в разумных пределах. Если задача действительно требует большого внимания, лучше создать отдельную задачу в трекере багов или задач.

**Часто используемые горячие клавиши**

Работа с кодом

* Ctrl + X — Вырезать
* Ctrl + C — Копировать
* Ctrl + V — Вставить
* Ctrl + Z — Отмена
* Ctrl + Y — Повторить отмененное действие
* Ctrl + A — Выделить всё

Редактирование кода

* Ctrl + K, Ctrl + C — Закомментировать выделенный код
* Ctrl + K, Ctrl + U — Раскомментировать выделенный код
* Ctrl + K, Ctrl + D — Форматирование кода (выравнивание)
* Ctrl + Space — Автодополнение
* Ctrl + Shift + V — Буфер обмена (просмотр предыдущих вставок)
* Alt + Up / Down — Перемещение строки вверх/вниз

Поиск и навигация

* Ctrl + F — Обычный поиск
* Ctrl + Shift + F — Поиск по всему проекту
* Ctrl + H — Замена
* Ctrl + Shift + H — Замена во всем проекте
* F12 — Перейти к определению
* Ctrl + T — Быстрый поиск по файлам, классам и методам
* Ctrl + G — Перейти к строке
* Ctrl + K, Ctrl + N / P — Следующий/предыдущий результат поиска

Отладка

* F5 — Запустить отладку
* Ctrl + F5 — Запуск без отладки
* Shift + F5 — Остановить отладку
* F9 — Установить/убрать точку останова
* F10 — Шаг с обходом (Step Over)
* F11 — Шаг с заходом (Step Into)
* Shift + F11 — Шаг с выходом (Step Out)