Что такое исключение?  
Исключение – это событие, возникающее из-за выполнения определенной команды, которая вызвала ошибку процессора

Что такое SEH?  
\*\*SEH (Structured Exception Handling)\*\* – это механизм Windows для обработки исключений на уровне системы. Он позволяет:

- Перехватывать аппаратные исключения (например, доступ к `NULL`-указателю).

- Обрабатывать программные ошибки.

- Выполнять очистку ресурсов при аварийном завершении.

Если возникает исключение, ядро перехватывает его и позволяет коду обработать исключение, если это возможно

SEH предоставляет две основные возможности: обработку завершения (termination handling) и обработку исключений (exception handling)

Основные ключевые слова:

- `\_\_try` – начало защищённого блока.

- `\_\_except` – блок обработки исключения (фильтр + обработчик).

- `\_\_finally` – блок завершения (выполняется всегда).

Что такое обработка завершений?  
Обработка завершений (termination handling) — блок finally, который обрабатывается всегда при возникновении исключения.

Применив обработчик завершения, мы не допустили преждевременного выполнения оператора **return**. Когда **return** пытается реализовать выход из блока **\_\_try**, компилятор проверяет, чтобы сначала был выполнен код в блоке **\_\_finally**, – причем до того, как оператору **return** в блоке **\_\_try** будет позволено реализовать выход из функции

Ключевое слово **\_\_leave** в блоке **\_\_try** вызывает переход в конец этого блока. Можно рассматривать это как переход на закрывающую фигурную скобку блока **\_\_try**.

И никаких неприятностей это не влечёт, потому что выход из блока **\_\_try** и вход в блок **\_\_finally** происходит **естественным** **образом**

Правда, нужно ввести дополнительную булеву переменную fFunctionOk, сообщающую о завершении функции: удачно оно или нет. Но это дает минимальные издержки

Разрабатывая функции, использующие обработчики завершения делайте именно так, инициализируйте все описатели ресурсов недопустимыми значениями перед входом в блок **\_\_try**. Тогда в блоке **\_\_finally** Вы проверите, какие ресурсы выделены успешно, и узнаете тем самым, какие из них следует потом освободить

Другой распространенный метод отслеживания ресурсов, подлежащих освобождению, – установка флага при успешном выделении ресурса. Код **\_\_finally** проверяет состояние флага и таким образом определяет, надо ли освобождать ресурс

Выполнение кода в блоке **\_\_finally** всегда начинается в результате возникновения одной из этих трех ситуаций. Чтобы определить, какая из них вызвала выполнение блока **\_\_finally**, вызовите встраиваемую функцию AbnormalTermination:

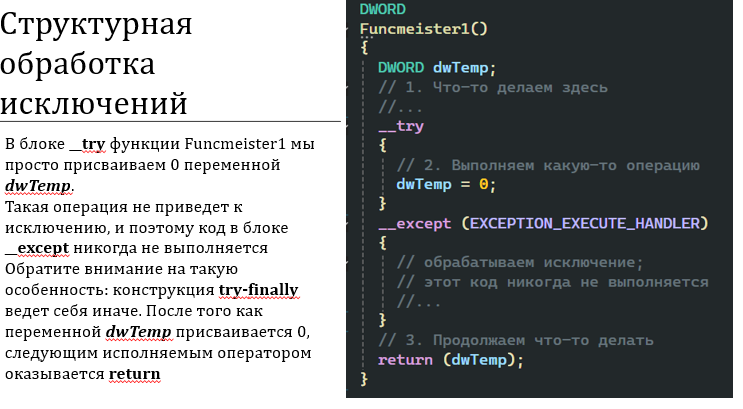
Её можно вызвать только из блока **\_\_finally**; она возвращает булево значение, которое сообщает, был ли преждевременный выход из блока **\_\_try**, связанного с данным блоком **\_\_finally**. Иначе говоря, если управление естественным образом передано из **\_\_try** в **\_\_finally**, **AbnormalTermination** возвращает FALSE. А если выход был преждевременным – то вызов **AbnormalTermination** дает TRUE



Обратите внимание на ключевое слово **\_\_except**. За блоком **\_\_try** всегда должен следовать либо блок **\_\_finally**, либо блок **\_\_except**

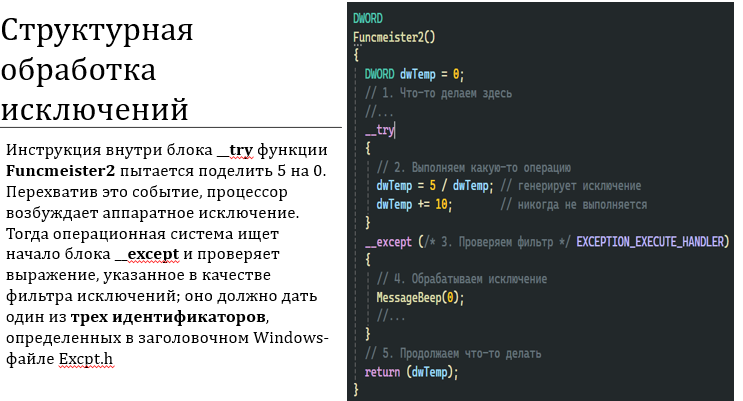
Для данного блока **\_\_try** нельзя указать одновременно и блок **\_\_finally**, и блок **\_\_except**; к тому же за **\_\_try** не может следовать несколько блоков **\_\_finally** или **\_\_except**

Однако **try-finally** можно вложить в **try-except**, и наоборот



Хотя ставить операторы **return**, **goto**, **continue** и **break** в блоке **\_\_try** обработчика завершения настоятельно не рекомендуется, их применение в этом блоке не приводит к снижению быстродействия кода или к увеличению его размера

Использование этих операторов в блоке **\_\_try**, связанном с блоком \_\_**except**, не вызовет таких неприятностей, как локальная раскрутка



Фильтры исключений:

* EXCEPTION\_EXECUTE\_HANDLER
* EXCEPTION\_CONTINUE\_SEARCH
* EXCEPTION\_CONTINUE\_EXECUTION

**EXCEPTION\_EXECUTE\_HANDLER** – это значение сообщает системе в основном вот что: «Я вижу это исключение; так и знал, что оно где-нибудь произойдет; у меня есть код для его обработки, и я хочу его сейчас выполнить.»

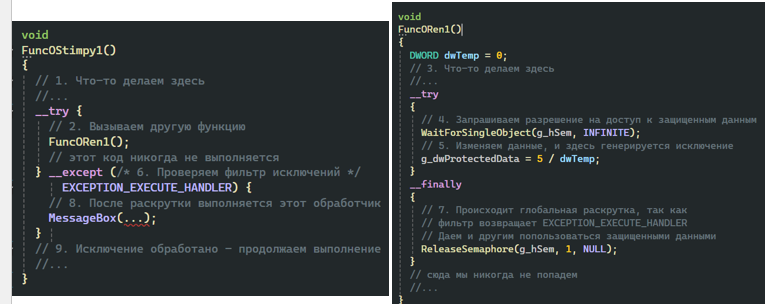
"Я знаю об этом исключении и хочу обработать его здесь."Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

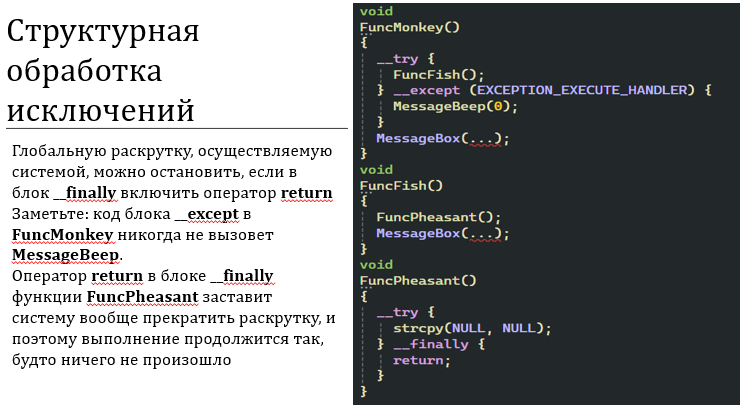
Содержимое, созданное искусственным интеллектом, может быть неверным.

В этот момент система проводит глобальную раскрутку, а затем управление передается коду внутри блока **\_\_except** (коду обработчика исключений). После его выполнения система считает исключение обработанным и разрешает программе продолжить работу

Приложение возобновляет выполнение с инструкции, следующей за блоком **\_\_except**. По окончании выполнения кода в блоке **\_\_except** управление передается на первую строку за этим блоком

Когда фильтр исключений возвращает **EXCEPTION\_EXECUTE\_HANDLER**, системе приходится проводить **глобальную раскрутку**. Она приводит к продолжению обработки всех незавершенных блоков **try-finally**, выполнение которых началось вслед за блоком **try-except**, обрабатывающим данное исключение





**EXCEPTION\_CONTINUE\_EXECUTION** – обнаружив такое значение выражения в фильтре, система возвращается к инструкции, вызвавшей исключение, и пытается выполнить ее снова (выполнение try снова)

"Притворимся, что ничего не случилось, и попробуем выполнить эту команду снова."

Если фильтр возвращает это, система **пытается продолжить выполнение с того места**, где произошло исключение.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**EXCEPTION\_CONTINUE\_SEARCH** – данный идентификатор указывает системе перейти к предыдущему блоку **\_\_try**, которому соответствует блок **\_\_except**, и обработать его фильтр (обработка фильтра предыдущего try)

"Я не хочу обрабатывать это исключение, пусть его попробует обработать другой блок выше по стеку."

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

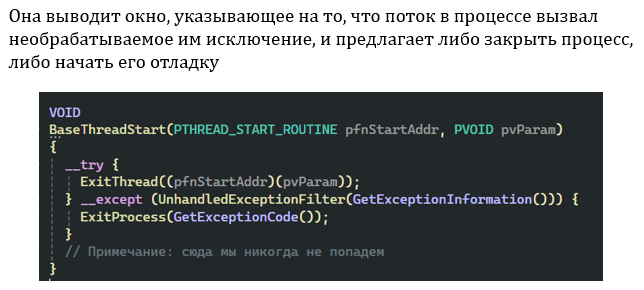
Это значит, что система пропускает при просмотре цепочки блоков любые блоки **\_\_try**, которым соответствуют блоки **\_\_finally** (а не **\_\_except**). Причина этого очевидна: в блоках **\_\_finally** нет фильтров исключений, а потому и проверять в них нечего

Встраиваемую функцию **GetExceptionCode** можно вызвать только из фильтра исключений (между скобками, которые следуют за **\_\_except**) или из обработчика исключений

Возбудить программное исключение несложно – достаточно вызвать функцию **RaiseException**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.



Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Что такое обработка исключений?

\*\*Обработка исключений\*\* – это механизм перехвата и реакции на исключительные ситуации. В SEH обработка исключений осуществляется через блоки *try и* except, а фильтры проверяют тип исключения и выполняют соответствующие действия.

- `EXCEPTION\_EXECUTE\_HANDLER` – выполнить обработчик.

- `EXCEPTION\_CONTINUE\_SEARCH` – передать исключение выше (найти фильтр).

- `EXCEPTION\_CONTINUE\_EXECUTION` – продолжить выполнение (если будет исключение оно будет проигнорировано).

Что такое защищенный блок кода?  
\*\*Защищённый блок кода\*\* – это блок, в котором возможны исключения, и для которого определены обработчики.

Для чего нужно ключевое слово \_\_leave?  
Ключевое слово *leave используется для выхода из блока* try, переходя к выполнению блока \_\_finally. Это позволяет корректно завершить выполнение функции без преждевременного выхода.

Что такое локальная и глобальная раскрутки?  
Локальная раскрутка происходит, когда управление передается в блок *finally из-за преждевременного выхода из блока* try (например, с использованием return). Глобальная раскрутка происходит, когда ошибка вызывает выход из нескольких уровней вложенных блоков *try, и выполнение кода возвращается в блоки* finally.

В чем главное отличие SEH от C++ исключений?  
Главное отличие заключается в том, что SEH — это механизм операционной системы, доступный для любых языков программирования, а C++ исключения являются частью самого языка. При этом компилятор MSVC использует SEH для реализации обработки исключений в C++. C++ исключения используют ключевые слова try, catch, и throw, в то время как SEH использует *try,* except, и \_\_finally.

Исключение – это событие, возникающее из-за выполнения определенной команды, которая вызвала ошибку процессора  
Исключения в некотором роде похожи на прерывания, основное отличие заключается в том, что исключение является **синхронным** и **технически воспроизводимым** при тех же условиях, в то время как прерывание является асинхронным и может произойти в любой момент

Примеры исключений: деление на ноль, точка останова, ошибка страницы, переполнение стека и недопустимая инструкция

\*\*SEH (Structured Exception Handling)\*\* – это механизм Windows для обработки исключений на уровне системы. Основная нагрузка по поддержке SEH ложится на компилятор, а не на ОС. Он позволяет:

- Перехватывать аппаратные исключения (например, доступ к `NULL`-указателю).

- Обрабатывать программные ошибки.

- Выполнять очистку ресурсов при аварийном завершении.

Если возникает исключение, ядро перехватывает его и позволяет коду обработать исключение, если это возможно

SEH предоставляет две основные возможности: обработку завершения (termination handling) и обработку исключений (exception handling)

Основные ключевые слова:

- `\_\_try` – начало защищённого блока (в котором могут возникать исключения).

- `\_\_except` – блок обработки исключения (фильтр + обработчик). Указывает, обработано ли исключение, и предоставляет код обработки, если это так

- `\_\_finally` – блок завершения (выполняется всегда). Предоставляет код, который гарантированно будет выполнен независимо от того, завершается ли блок \_\_try обычным образом, с помощью инструкции return или из-за исключения

- `\_\_leave` – предоставляет оптимизированный механизм для перехода к блоку \_\_finally откуда-либо из блока \_\_try

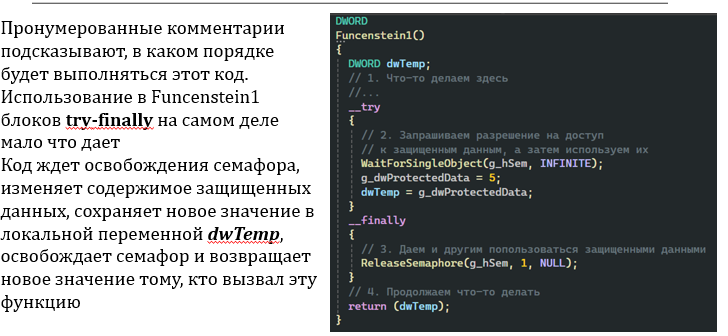
Хотя всю работу по отлову исключений берёт на себя операционная система, однако основная нагрузка по поддержке SEH ложится на компилятор, а не на операционную систему

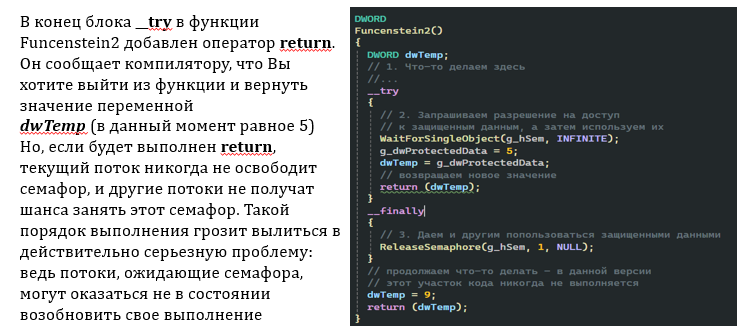
Он генерирует специальный код на входах и выходах **блоков исключений** (**exception blocks**), создает таблицы вспомогательных структур данных для поддержки SEH и предоставляет функции обратного вызова, к которым система могла бы обращаться для прохода по блокам исключений

Компилятор отвечает и за формирование **стековых фреймов** (**stack frames**) и другой внутренней информации, используемой ОС. **Стековым фреймом** называется область стека, которую занимают локальные объекты одного блока

Данные понятия нам понадобятся далее при обсуждении понятия раскрутки стека!

Защищенный блок кода (protected code block) — это блок, ограниченный оператором \_\_try, в котором предполагается, что могут возникнуть исключения, которые необходимо обработать.





Применив обработчик завершения, мы не допустили преждевременного выполнения оператора **return**. Когда **return** пытается реализовать выход из блока **\_\_try**, компилятор проверяет, чтобы сначала был выполнен код в блоке **\_\_finally**, – причем до того, как оператору **return** в блоке **\_\_try** будет позволено реализовать выход из функции

Вызов ReleaseSemaphore в обработчике завершения (в функции Funcenstein2) гарантирует освобождение семафора – поток не сможет случайно сохранить права на семафор и тем самым лишить процессорного времени все ожидающие этот семафор потоки

После выполнения блока **\_\_finally** функция фактически завершает работу. Любой код за блоком **\_\_finally** не выполняется, поскольку возврат из функции происходит внутри блока **\_\_try**

Каким же образом компилятор гарантирует выполнение блока **\_\_finally** до выхода из блока **\_\_try**?

Дело вот в чем:

Просматривая исходный текст, компилятор видит, что Мы вставили **return** внутрь блока **\_\_try**. Тогда он генерирует код, который сохраняет возвращаемое значение (в нашем примере 5) в созданной им же временной переменной. Затем создает код для выполнения инструкций, содержащихся внутри блока **\_\_finally**, – это называется **локальной раскруткой** (**local unwind**)

По сути раскрутка это процесс освобождения локальных объектов каждого из блоков из стека процесса (в частности вложенных блоков)

Точнее, **локальная раскрутка** происходит, когда система выполняет блок **\_\_finally** из-за **преждевременного** выхода из блока **\_\_try**. Значение временной переменной, сгенерированной компилятором, возвращается из функции после выполнения инструкций в блоке **\_\_finally**

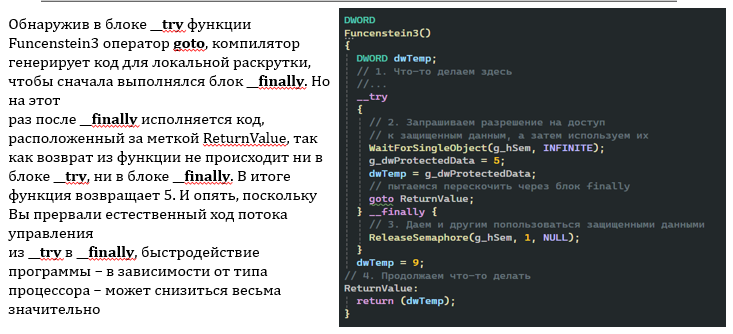
Как видите, чтобы все это вытянуть, компилятору приходится генерировать дополнительный код, а системе – выполнять дополнительную работу. На разных типах процессоров поддержка обработчиков завершения реализуется по-разному, вплоть до **сотен тысяч дополнительных** машинных команд, что может отрицательно сказаться на быстродействии программы

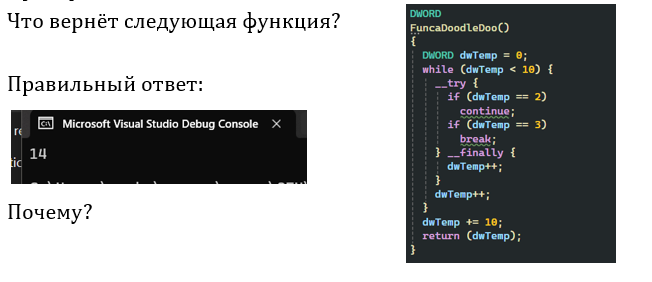
Поэтому лучше не писать код, вызывающий преждевременный выход из блока \_\_**try** обработчика завершения

Обработка исключений предназначена для перехвата тех исключений, которые происходят не слишком часто (в нашем случае – преждевременного возврата)

Если же какое-то исключение – чуть ли не норма, гораздо эффективнее проверять его явно, не полагаясь на SEH.

Заметьте: когда поток управления выходит из блока **\_\_try** естественным образом (как в Funcenstein1), издержки от вызова блока **\_\_finally** минимальны, так как для входа в **\_\_finally** при **нормальном** выходе из **\_\_try** исполняется ***всего одна*** машинная команда – вряд ли Вы заметите ее влияние на быстродействие своей программы





Итак, **обработчики завершения**, хоть и весьма эффективные однако при преждевременном выходе из блока **\_\_try**, могут дать **нежелательные** результаты именно потому, что предотвращают досрочный выход из блока **\_\_try**

Лучше всего избегать любых операторов, способных вызвать преждевременный выход из блока **\_\_try** обработчика завершения. А в идеале – удалить все операторы **return**, **continue**, **break**, **goto** (и им подобные) как из блоков **\_\_try**, так и из блоков **\_\_finally**