Шангин Леонид

Я выполнил рефакторинг кода и поместил результаты в отдельном Github репозитории, ссылку на который прикрепил ниже.

Буду рад, если старания не окажутся напрасными и Вы просмотрите итог проделанной работы.

https://github.com/Shangin-Leonid/MyOffice\_code\_review\_task

**Существенные изменения**

|  |  |  |
| --- | --- | --- |
| **№ строки** | **Описание проблемы, способ решения** | **Критичность, комментарии** |
| 68  72 | Исправлена ошибка выхода за границы массива:  "i <= 16" -> "i < 16"  "[16]" -> "[15]" |  |
| 87-88 | clearKey() вызывается в деструкторе ~FileEncryptor(), поэтому следует избавиться от выбрасывания исключения. | Вообще непонятно, к чему там была проверка на nullptr: "delete nullptr" корректно. |
| 75  85-91 | clearKey(), вызываемая в деструкторе ~FileEncryptor(), выбрасывает исключение. Недопустимо! Сделано noexcept. |  |
| 142  154-156  160-162 | Воспользуемся умным указателем вместо классического и забудем про возможные утечки. |  |
| 142-143  5  115-128  154-156  160-162 | Предпочтём std::unique\_ptr классическому указателю, сняв с себя ответственность за владение ресурсом FileProcessorFactorySingleton и предотвратив возможные утечки памяти. |  |
| 8 | Для FileProcessor добавлен необходимый виртуальный деструктор. |  |
| 25  45  78 | Для файловых потоков добавлены проверки на успешность окрытия. | Вот только как именно реагировать на ошибки, я не знаю. |
| 10-12 | FileProcessor позволяет наследоваться от него, не переопределяя метод processFile().  Код в таком виде сулит неожиданные выбросы исключений и падения там, где мы их не ждём.  Для подобных целей C++ предусмотрел абстрактные базовые классы. Сделаем злосчастный метод чистым виртуальным. |  |
| 18 | Судя по названию и логике использования, IdentityFileProcessor предполагался как разновидность FileProcessor. Тогда:  "encodeFile(std::string fileContent)" -> "processFile(const std::string & filePath)" |  |
| 15  18  21  23  36  44  63  77  85 | Для конечных классов, наследованных от абстрактных, добавлен final, снят virtual с методов и добавлен для переопределённых функций override. |  |
| 41-42 | Реализованы недостающие методы в FileCompressor.  Они дополнены соответствующими noexcept спецификаторами. Для этого создан псевдоним типа параметров - членов класса. | Разработчик, оставивший данные методы нереализованными, должен был поставить заглушку: "static\_assert(false, "FIXME"); |
| 18 | Если мы используем C++17, то обозначить параметр, как неиспользуемый, чтобы избежать предупреждений компилятора:  "std::string filePath" -> "[[maybe\_unused]] std::string filePath" | Если стандарт ниже 17-ого, но используется GCC или CLang - "\_\_attribute\_\_((unused))" |
| 105 - ... | Перенёс переменную instance внутрь метода getInstance, который теперь возвращает указатель. Я не силён в многопоточности, но знаю, что классический синглтон Майерса, начиная с C++11, является потокобезопасным и реализуется именно так. | Кажется, не стоит тратить время на осознание этих тонкостей - в таком вопросе можно довериться опыту гуру. |
| 67 | Следует ли поймать возможное исключение и как-то отреагировать на него? | Или такие штуки будем ловить не здесь, а дальше? |

**Логика работы программы**

|  |  |  |
| --- | --- | --- |
| **№ строки** | **Описание проблемы, способ решения** | **Критичность, комментарии** |
| 116-127 | Нужно обработать случай несовпадения ни с одним из предусмотренных значений mode:  "if ... if ... if" -> "if ... else if ... else" |  |

**Влияющие только на эффективность кода (память, время)**

|  |  |  |
| --- | --- | --- |
| **№ строки** | **Описание проблемы, способ решения** | **Критичность, комментарии** |
| 10  18  23  31  38  44  52  77  93  115 | Передача параметров (строк) по ссылке, а не по значению. |  |
| 4  145-152 | Можно избавиться от std::vector, убрав соответствующий заголовок, сократив затраты дополнительной памяти, время работы программы и время компиляции. |  |
| 58  39  54 | Мне непонятно наличие поля compressionParams в FileCompressor. Предлагаю обойтись без него. |  |
| 68 | Авось ускорит работу программы :-)  "i++" -> "++i" |  |

**Культура и гигиена кода**

|  |  |  |
| --- | --- | --- |
| **№ строки** | **Описание проблемы, способ решения** | **Критичность, комментарии** |
|  | Форматирование, имена переменных (вкусовщина). Тут надо соответствовать стандартам, принятым в команде. В условии они не даны. |  |
| 68  70 | Не помешало бы магической константе 16 присвоить имя.  А 0x20 - какой-то определённый символ? Пробел? Тогда давайте прибавлять его:  " + 0x20 " -> " + ' ' " |  |
| 116-127 | Предусмотренные значения mode можно оформить в виде констант, #define макросов или даже обратиться к "enum class : int", отойдя от строк и затрат на работу с ними. |  |
|  | Все пустые выражения в объявлениях и определениях заменены на void и /\* empty \*/ (кроме аргументов деструкторов). |  |
| 1-6 | "#include "\*"" -> "#include <\*>"  "time.h" -> "<ctime>" |  |
| 163  87 | "Sometihing" -> "Something"  "!encryptionKey" -> "\_encryptionKey == nullptr"  Добавлены переводы строк в различных потоковых выводах. |  |
|  | "noexcept(false)" я бы убрал. Обязательной с точки зрения языка такая запись не является, ведь она равносильна своему отсутствию. Однако загромождает код. | Меньше слов => плотнее контекст => легче удерживается в голове => более читаемый и поддерживаемый код. |

**Предложения (возможно выходят за рамки ревью)**

|  |
| --- |
| **Описание проблемы, способ решения** |
| Вынести объявления классов в другой файл. Но это уже архитектура более высокого уровня. |
| Если некоторые классы используются только в данной единице трансляции, то стоит их обернуть в безымянное пространство имён. Но подозреваю, что в настоящем коде все объявления классов будут вынесены в другие файлы. |
| Можно добавить noexcept для всех функций, не вызывающих исключений. Хотя, наверное, в задании предполагается, что они внутри имеют более сложную логику и не позволяют это сделать. |
| Пожалуй, лучше в качестве исключений выбрасывать не строки, а какие-то собственные классы исключений, наследующиеся от стандартных. |
| К чему на 102 строчке "static\_assert(std::is\_nothrow\_move\_constructible<FileEncryptor>..." ?  Чтобы код нельзя было компилировать без реализации move-конструктора? Тогда лучше "static\_assert(false, "FileEncryptor need move constructor")".  Чтобы гарантировать, что move конструктор, пусть и дефолтный, есть? Тогда ладно. Но почему проверка только для данного класса?  Есть "правило пяти". Возможно, в каждом из классов стоит реализовать все 5 обязательных методов. |
| В языке C предусмотрены функции для парсинга параметров командной строки. Для получения значения mode можно воспользоваться <unistd.h> или <getopt.h>.  Такая техника позволила бы при усложнении значений параметров легче переписать код на новый лад, не занимаясь разработкой собственных функций для парсинга и анализа пользовательского ввода. |