

# Онлайн образование

[otus.ru](https://otus.ru)



Проверить, идет ли запись

# Меня хорошо видно && слышно?



Тема вебинара

Разработчик C++ - базовый курс

# Работа с библиотеками вручную



**Пальчуковский Евгений**

Разработчик ПО

Развиваю технологии финансовых услуг с помощью C++

Telegram: @palchukovsky

Email: eugene@palchukovsky.com

# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе в  
Telegram



Задаем  
вопрос в чат



Вопросы вижу в чате,  
могу ответить не сразу

# Цели вебинара

1. Вспомним, что такое библиотеки
2. Вспомним, какие библиотеки бывают - статические, динамические
3. Разберём, как использовать их вручную, с примерами в CMake
4. Посмотрим, что там “под капотом” у механизма использования библиотек

# Мир C++ строится из библиотек, и наоборот



# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества



# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества
- Работа на более **высоком уровне абстракции**  
... мыслим категориями бизнес задач, **не зарываемся в код**





# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества
- Работа на более **высоком уровне абстракции**  
... мыслим категориями бизнес задач, **не зарываемся в код**
- **Надёжность:** У сообщества больше ресурсов поддерживать



# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества
- Работа на более **высоком уровне абстракции**  
... мыслим категориями бизнес задач, **не зарываемся в код**
- **Надёжность:** У сообщества больше ресурсов поддерживать
- **Сторонняя экспертиза:** Серьезные дяди и тёти уже все написали



# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества
- Работа на более **высоком уровне абстракции**  
... мыслим категориями бизнес задач, **не зарываемся в код**
- **Надёжность:** У сообщества больше ресурсов поддерживать
- **Сторонняя экспертиза:** Серьезные дяди и тёти уже все написали

## Библиотеки проекта

- **Разделение ответственностей:** Каждой компоненте - по библиотеке!



# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества
- Работа на более **высоком уровне абстракции**  
... мыслим категориями бизнес задач, **не зарываемся в код**
- **Надёжность:** У сообщества больше ресурсов поддерживать
- **Сторонняя экспертиза:** Серьезные дяди и тёти уже все написали

## Библиотеки проекта

- **Разделение ответственностей:** Каждой компоненте - по библиотеке!
- **Модули и плагины:** Пусть клиенты делают сами



# Ценность библиотек

## Чужие библиотеки

- **Экономим:** Используем результаты работы сообщества
- Работа на более **высоком уровне абстракции**  
... мыслим категориями бизнес задач, **не зарываемся в код**
- **Надёжность:** У сообщества больше ресурсов поддерживать
- **Сторонняя экспертиза:** Серьезные дяди и тёти уже все написали

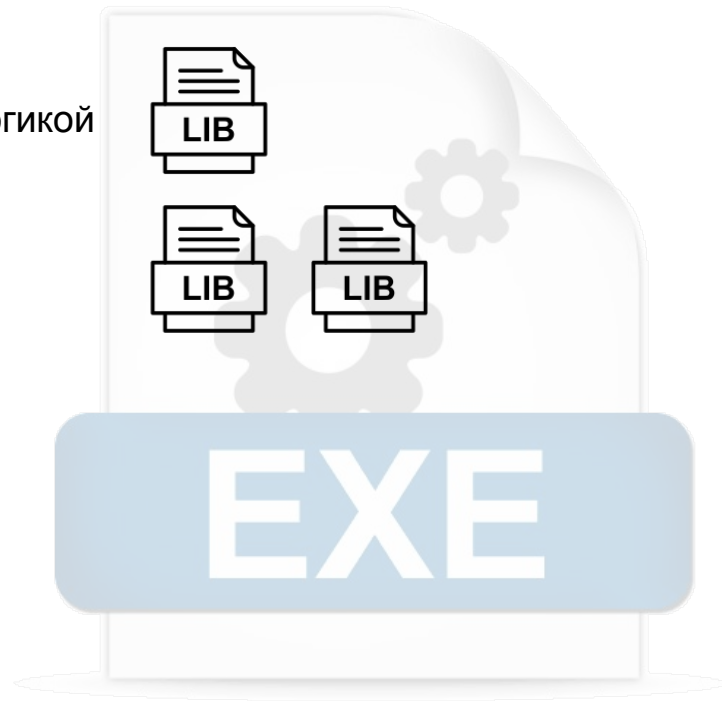
## Библиотеки проекта

- **Разделение ответственностей:** Каждой компоненте - по библиотеке!
- **Модули и плагины:** Пусть клиенты делают сами
- **Еще раз экономим:** Переиспользуем для других проектов



# Статическая библиотека (static library)

1. Берём несколько единиц трансляции с необходимой логикой



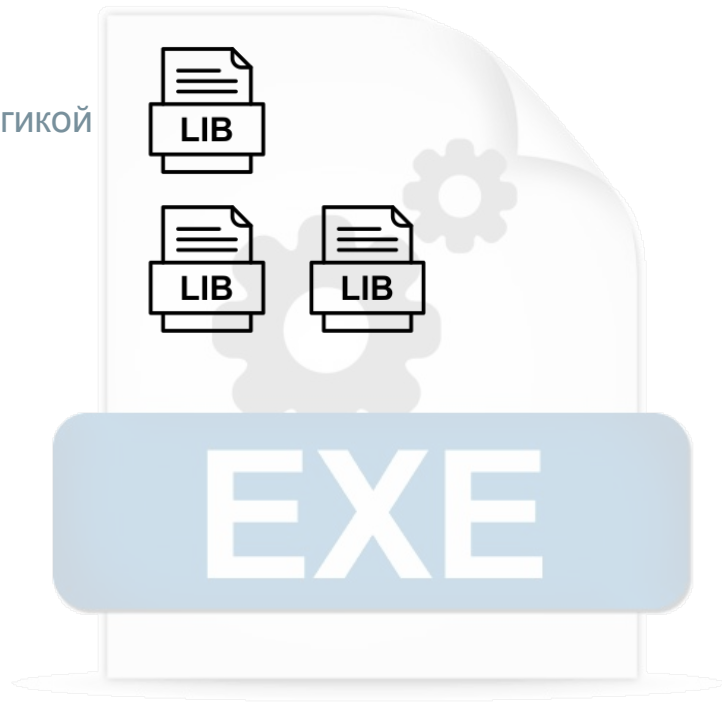
# Статическая библиотека (static library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных модулей



# Статическая библиотека (static library)

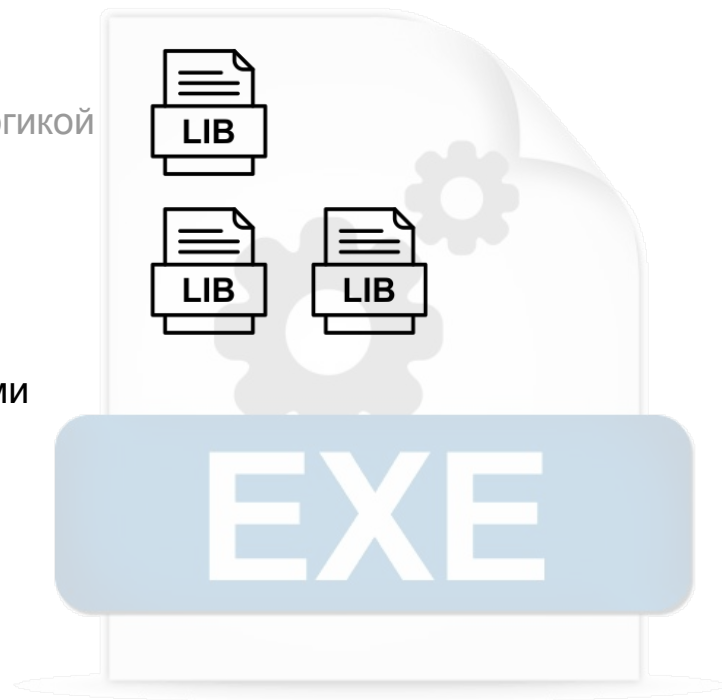
1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных модулей
3. Архивируем объектные файлы как-нибудь





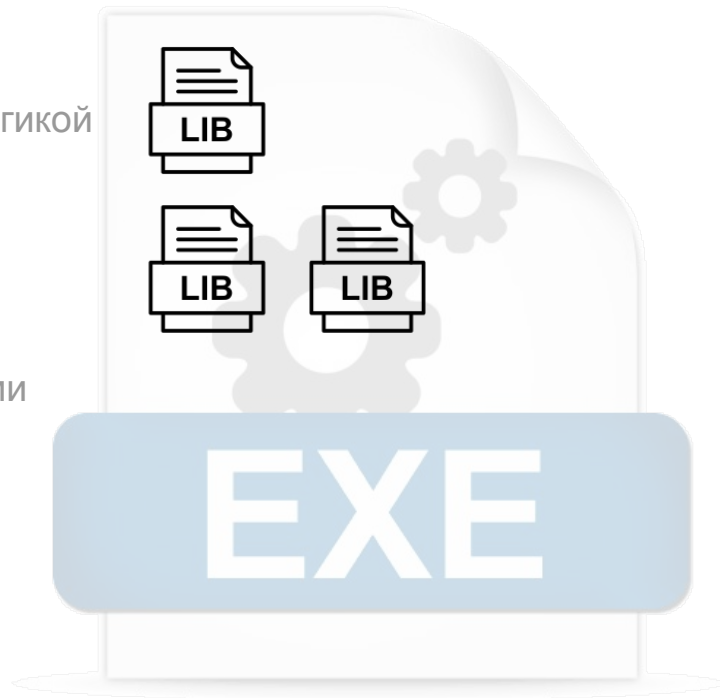
# Статическая библиотека (static library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных модулей
3. Архивируем объектные файлы как-нибудь
4. Добавляем набор заголовочных файлов с объявлениями



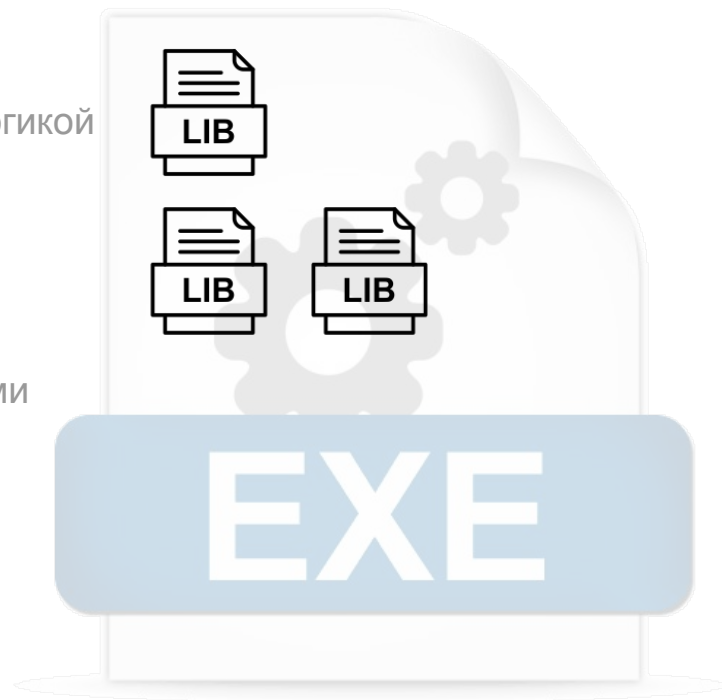
# Статическая библиотека (static library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных модулей
3. Архивируем объектные файлы как-нибудь
4. Добавляем набор заголовочных файлов с объявлениями
5. Отдаём всем, кому нужен наш код



# Статическая библиотека (static library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных модулей
3. Архивируем объектные файлы как-нибудь
4. Добавляем набор заголовочных файлов с объявлениями
5. Отдаём всем, кому нужен наш код
6. Код будет добавлен в каждый исполняемый файл



# Статическая библиотека (static library)

Особенности использования:

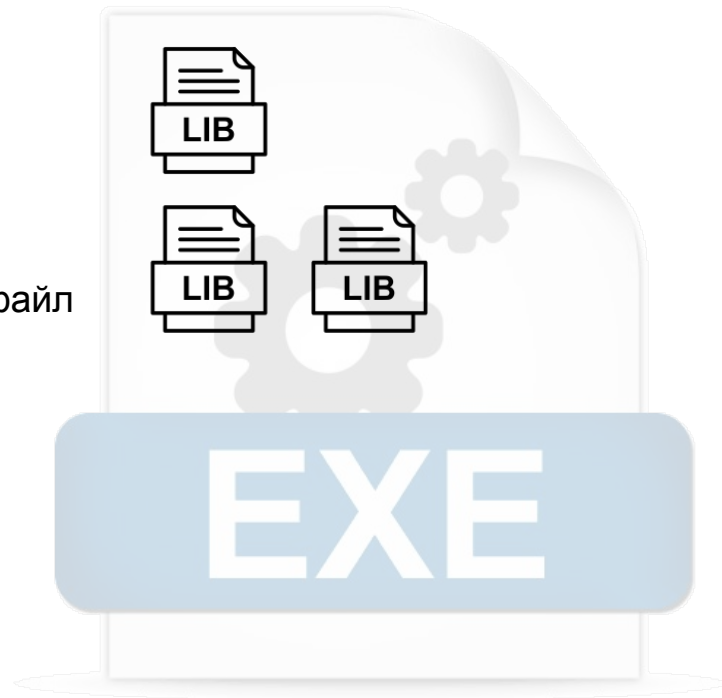
- Простое использование - подключил и забыл



# Статическая библиотека (static library)

Особенности использования:

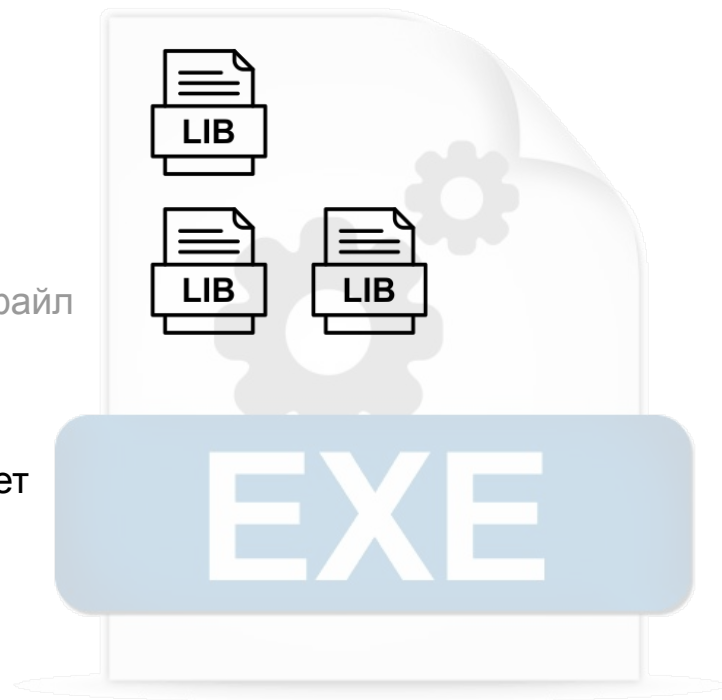
- Простое использование - подключил и забыл
- ... но если что-то нужно поправить - пересобирай exe-файл



# Статическая библиотека (static library)

Особенности использования:

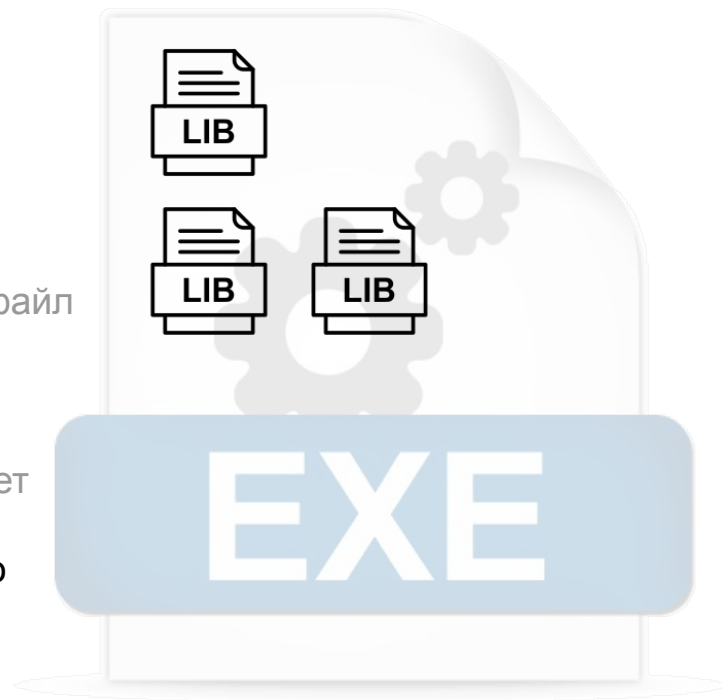
- Простое использование - подключил и забыл
- ... но если что-то нужно поправить - пересобирай exe-файл
- Дублирование кода библиотеки
- ... в каждом исполняемом файле, которые её использует



# Статическая библиотека (static library)

Особенности использования:

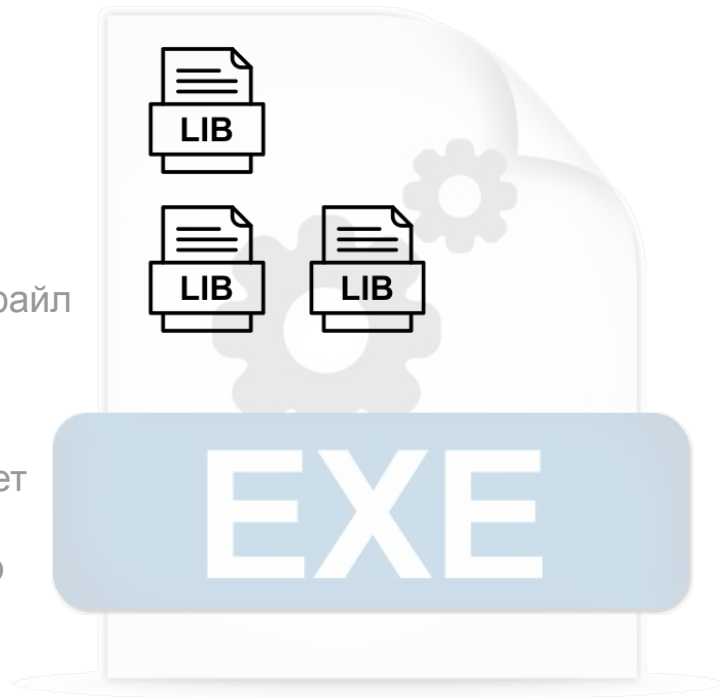
- Простое использование - подключил и забыл
- ... но если что-то нужно поправить - пересобирай exe-файл
- Дублирование кода библиотеки
- ... в каждом исполняемом файле, которые её использует
- ... но есть оптимизация исключения неиспользованного



# Статическая библиотека (static library)

## Особенности использования:

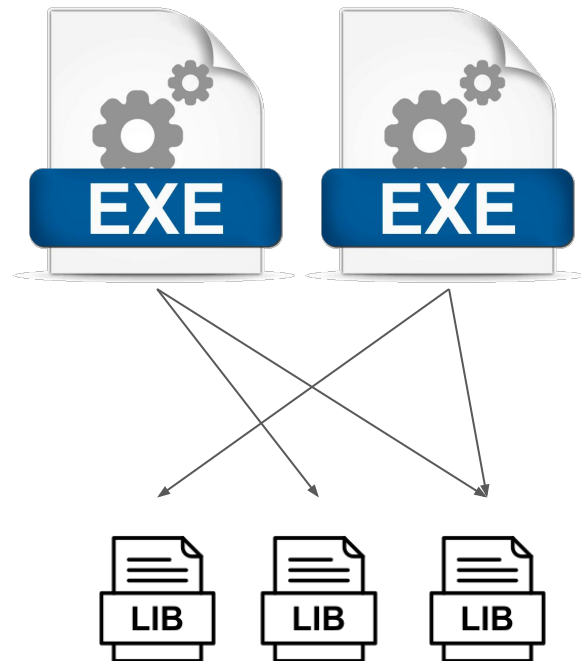
- Простое использование - подключил и забыл
- ... но если что-то нужно поправить - пересобирай exe-файл
- Дублирование кода библиотеки
- ... в каждом исполняемом файле, которые её использует
- ... но есть оптимизация исключения неиспользованного
- Не включает свои зависимости ❌
- Зависимости разрешают в linking time ✅





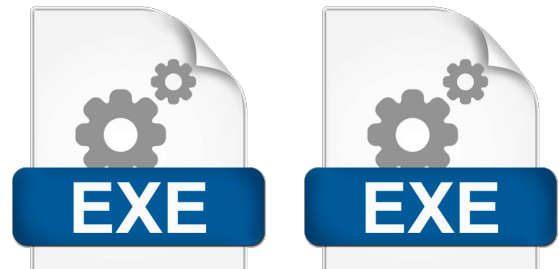
# Динамическая библиотека (dynamic library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных файлов
3. Линкуем всё вместе
4. Фактически, получая готовый к исполнению код



# Динамическая библиотека (dynamic library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных файлов
3. Линкуем всё вместе
4. Фактически, получая готовый к



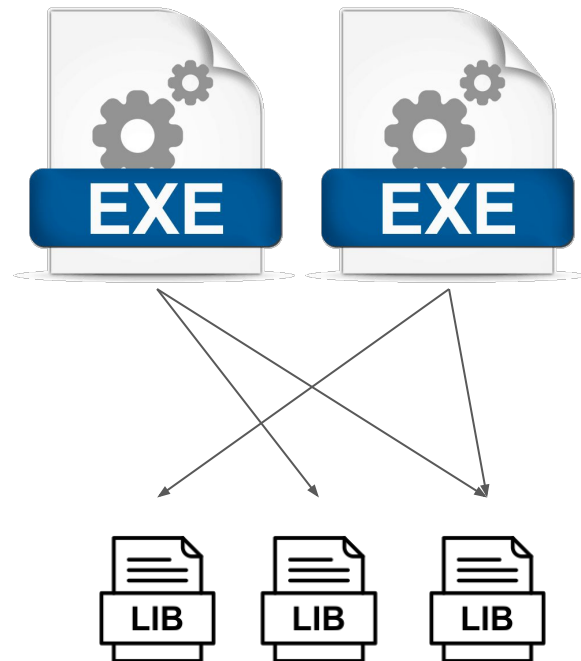
## Статическая библиотека (static library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных модулей
3. Архивируем объектные файлы как-нибудь



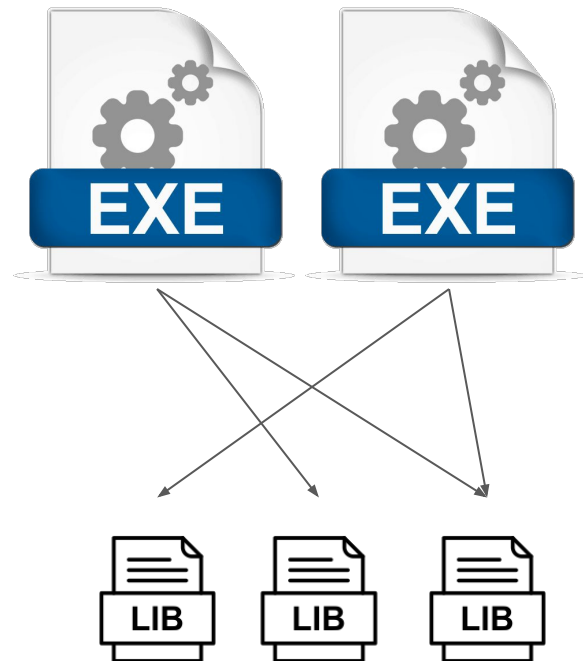
# Динамическая библиотека (dynamic library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных файлов
3. Линкуем всё вместе
4. **Фактически, получая готовый к исполнению код**



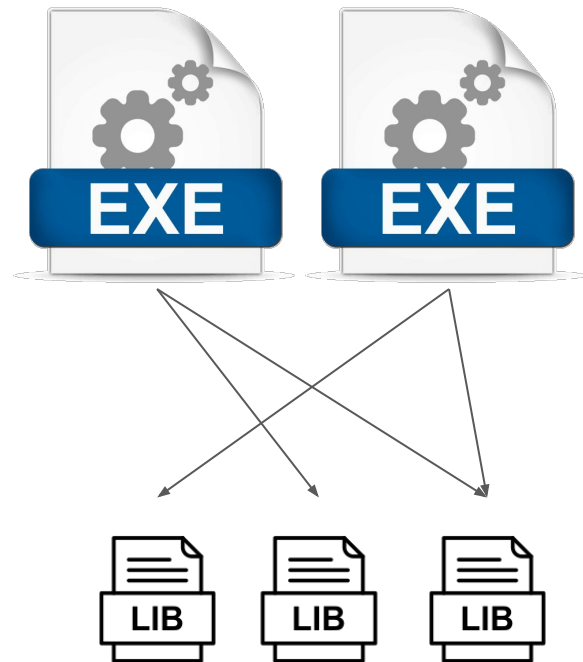
# Динамическая библиотека (dynamic library)

1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных файлов
3. Линкуем всё вместе
4. Фактически, получая готовый к исполнению код
5. Можно добавить заголовочные файлы, **но не обязательно**
6. Отдаём всем, кому нужен наш код





# Динамическая библиотека (dynamic library)

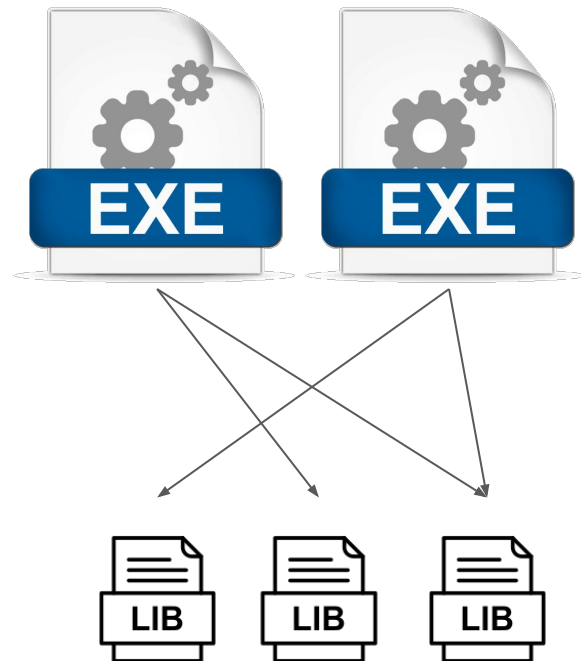
1. Берём несколько единиц трансляции с необходимой логикой
2. Делаем из них несколько объектных файлов - ассемблируем и линкуем
3. Фактически, получая готовый к исполнению код
4. Можно добавить заголовочные файлы, но не обязательно
5. Отдаём всем, кому нужен наш код
6. Код будет добавлен лежать в одном месте
7. ... а использоваться, потенциально, в нескольких проектах



# Динамическая библиотека (dynamic library)

Особенности использования:

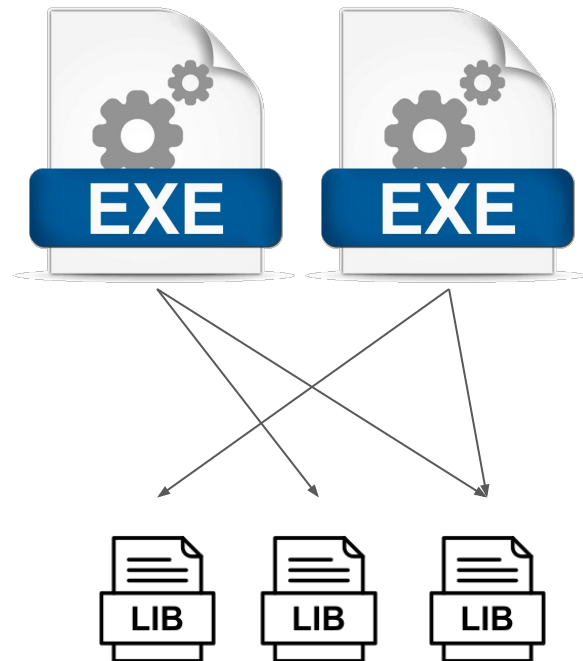
- Немного сложное использование 
- Разделяемое использование 



# Динамическая библиотека (dynamic library)




Особенности использования:

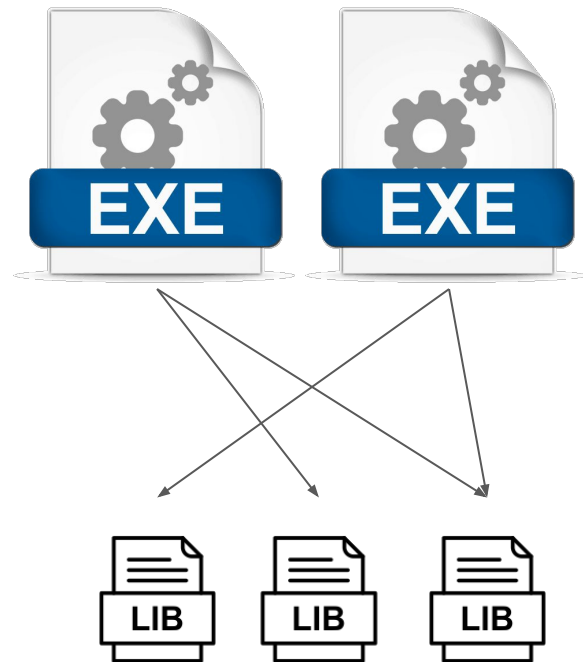
- Немного сложное использование
- Разделяемое использование
- Можно подключать/выгружать динамически ✓ ✓ ✓



# Динамическая библиотека (dynamic library)

Особенности использования:

- Немного сложное использование
- Разделяемое использование
- Можно подключать/выгружать динамически
- Включает все статические зависимости 
- Динамические зависимости **разрешаются в runtime**  

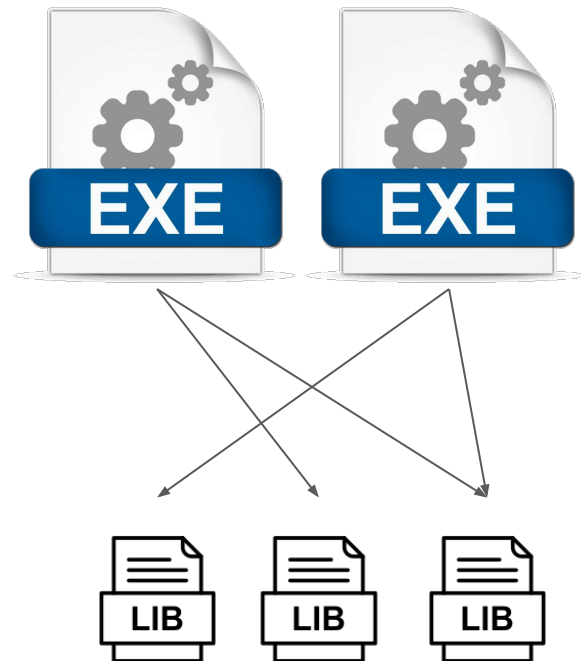




# Динамическая библиотека (dynamic library)

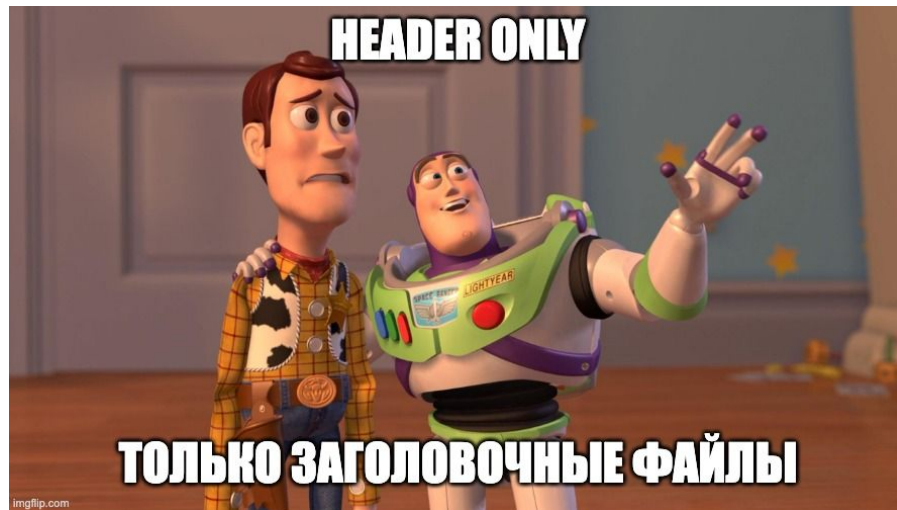
Особенности использования:

- Немного сложное использование
- Разделяемое использование
- Можно подключать/выгружать динамически
- Включает все статические зависимости
- Динамические зависимости разрешаются в runtime
- Если что - можно обновить только библиотеку
- ... но только, если её интерфейс не поменялся



# Header-only

- Содержит только заголовочные файлы
- Подключается обычным “#include”
- Компилируется каждый раз заново
- Больше путей оптимизации компилятором



# Примеры из жизни

**Boost (C++ libraries):** Библиотека классов C++ для многих случаев.

Не полностью, но большей частью состоит только из заголовков. Их достаточно распаковать и сделать **#include**.



**GoogleTest:** Удобнее “статически”,

особенно, если тесты нужно запустить на другом компьютере.



**GUI, компьютерная графика и медиа:** Удобнее “динамически” - одни для всех,

большой размер, долгая установка в систему, уже, скорее всего, есть на компьютере.



см. “static\_lib/lib” в приложенном  
примере

# Статическая библиотека

sum.h

1

```
#pragma once  
  
namespace library {  
  
    int sum(int a, int b);  
  
} // namespace library
```

CMakeLists.txt

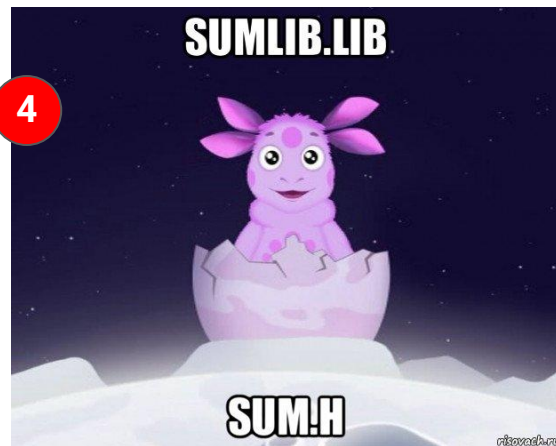
3

```
cmake_minimum_required(VERSION 3.5)  
  
project(static_library)  
  
add_library(sumLib STATIC  
    sum.h  
    sum.cpp  
)
```

sum.cpp

2

```
#include "sum.h"  
  
namespace library {  
  
    int sum(const int a, const int b) {  
        return a + b;  
    }  
  
} // namespace library
```



# Статическая библиотека

Windows, MS Visual C++

Утилита пакета VS **DUMPBIN**

Команда “список символов в  
sum.obj”

```
1. > dumpbin /SYMBOLS sum.obj
2. ...
3. 009 00000000 SECT3 notype () External | ?sum@library@@YAHNN@Z (int __cdecl library::sum(int,int))
4. ...
```

# Статическая библиотека

Windows, MS Visual C++

Утилита пакета VS **DUMPBIN**

Команда "список символов в  
sum.obj"

```
1. > dumpbin /SYMBOLS sum.obj
2. ...
3. 009 00000000 SECT3 notype () External | ?sum@library@@YAHNN@Z {int __cdecl library::sum(int,int)}
4. ...
```

Запись #9 в секции  
SECT3

Что за функция -  
закодировано

Расшифровка от  
DUMPBIN

# Статическая библиотека

## Подключение в Windows, MS Visual C++

Что внутри main.obj  
исполняемого файла?

```
1. > dumpbin /SYMBOLS main.obj
2. ...
3. 04C 00000000 UNDEF  notype ()      External      | ?sum@library@@YAHHH@Z (int __cdecl library::sum(int,int))
4. ...
5. 062 00000000 SECT4  notype ()      External      | main
```

# Статическая библиотека

## Подключение в Windows, MS Visual C++

Что внутри main.obj  
исполняемого файлы?

```
1. > dumpbin /SYMBOLS main.obj
2. ...
3. 04C 00000000 UNDEF  notype ()      External      | ?sum@library@@YAHNN@Z (int __cdecl library::sum(int,int))
4. ...
5. 062 00000000 SECT4  notype ()      External      | main
```

Корневая функция main - локальная,  
старт исполняемого файла



см. "static\_lib/lib" в приложенном примере

# Статическая библиотека

Подключение в Windows, MS Visual C++

Что внутри main.obj  
исполняемого файлы?

Функция "не определена", её  
придётся искать во время  
линковки

Это и есть наша функция из  
нашей библиотеки

```
1. > dumpbin /SYMBOLS main.obj
2. ...
3. 04C 00000000 UNDEF notype () External | ?sum@library@@YAHNN@Z (int __cdecl library::sum(int,int))
4. ...
5. 062 00000000 SECT4 notype () External | main
```

Корневая функция main - локальная,  
старт исполняемого файла

# Статическая библиотека

Linux, GCC

Системная утилита  
**nm** (name mangling)

Команда “список символов в  
libsumLib.a”

```
1. > nm libsumLib.a
2. ...
3. 0000000000000000 T __ZN7library3sumEii
4. ...
```

# Статическая библиотека

Linux, GCC

Системная утилита  
**nm** (name mangling)

Команда “список символов в  
libsumLib.a”

```
1. > nm libsumLib.a
2. ...
3. 0000000000000000 T __ZN7library3sumEii
4. ...
```

Что за функция - закодировано

# Статическая библиотека

## Подключение в Linux, GCC

Что внутри main.obj  
исполняемого файлы?

Наша функция из нашей библиотеки.

Функция "не определена" (U),  
её придётся искать во время линковки.

```
1. > nm main.cpp.o
2. ...
3.          U __ZN7library3sumEii
4. ...
5. 0000000000000000 T _main
```

Корневая функция main - локальная,  
старт исполняемого файла

# Декорирование имён

## Name mangling (найм мАнгилинг)

Процесс изменения имён функций, структур и классов компилятором с добавлением в имена дополнительной информации.

- наследие C
- линкер использует старый подход из C, чтобы поддержать уже созданное
- позволяет работать перегрузке
- даёт возможность указать соглашение о вызове
- является деталью реализации конкретного набора инструментов
- нет общих стандартов



# Декорирование имён

Name mangling (найм мАнгилинг)

Если указывать только имя  
сущности, то как линковщик:

- различит функции, если они перегружены по типу аргументов?
- найдёт функцию в namespace?
- отличит константный метод от не константного?

```
#pragma once
```

```
namespace library {
```

```
int sum(int a, int b);
```

```
double sum(double a, double b);
```

```
class A {
```

```
void Do();
```

```
void Do() const;
```

```
};
```

```
} // namespace library
```

# Декорирование имён

## Windows, MS Visual C++

```
1. > dumpbin /SYMBOLS sum.obj
2. ...
3. 009 00000000 SECT3  notype ()      External      | ?sum@library@@YAHNN@Z (int __cdecl library::sum(int,int))
4. ...
```

## Linux, GCC

```
1. > nm libsumLib.a
2. ...
3. 0000000000000000 T __ZN7library3sumEii
4. ...
```

см. “static\_lib/lib” в приложенном примере

# Декорирование имён

## Windows, MS Visual C++

```
1. > dumpbin /SYMBOLS sum.obj
2. ...
3. 009 00000000 SECT3 notype () External | ?sum@library@@YAHNN@Z (int __cdecl library::sum(int,int))
4. ...
```

**H** - код типа **int**. Порядок: возвращаемый тип, 1-ый аргумент, 2-ой аргумент.

Имя namespace

**i** - код типа **int**. Порядок: 1-ый аргумент, 2-ой аргумент.

## Linux, GCC

```
1. > nm libsumLib.a
2. ...
3. 0000000000000000 T __ZN7library3sumEii
4. ...
```



# Динамическая библиотека

Linux, GCC

см. "dynamic\_lib/lib" в  
приложенном примере

1 sum.h

```
#pragma once

namespace library {

    int sum(int a, int b);

} // namespace library
```

CMakeLists.txt

3

```
cmake_minimum_required(VERSION 3.5)

project(dynamic_library)

add_library(dynamicLib SHARED
    sum.h
    sum.cpp
)
```

Это лишь разница  
"STATIC" → "SHARED"?

sum.cpp

2

```
#include "sum.h"

namespace library {

    int sum(const int a, const int b) {
        return a + b;
    }

} // namespace library
```

4



# Динамическая библиотека

## Linux, GCC

1. `> nm libsumLib.so`

```
2. 000000000201020 B __bss_start
3. 000000000201020 b completed.6985
4.                w __cxa_finalize@@GLIBC_2.2.5
5. 0000000000000570 t deregister_tm_clones
6. 00000000000005e0 t __do_global_dtors_aux
7. 000000000200dd0 t __do_global_dtors_aux_fini_array_entry
8. 000000000200dd8 d __dso_handle
9. 000000000200de0 d _DYNAMIC
10. 000000000201020 D _edata
11. 00000000000201028 B __end
12. 0000000000000658 T _fini
13. 0000000000000620 t frame_dummy
14. 000000000200dc8 t __frame_dummy_init_array_entry
15. 0000000000000708 r __FRAME_END__
16. 000000000201000 d _GLOBAL_OFFSET_TABLE_
17.                w __gmon_start__
18. 0000000000000664 r __GNU_EH_FRAME_HDR
19. 0000000000000538 T _init
20.                w _ITM_deregisterTMCloneTable
21.                w _ITM_registerTMCloneTable
22. 00000000000005a0 t register_tm_clones
23. 000000000201020 d __TMC_END__
24. 000000000000063e T _ZN7library3sumEdd
25. 000000000000062a T _ZN7library3sumEii
```

Полноценный исполняемый  
файл.

Компилятор включил в него  
всё что нужно для такого  
файла.

Файл библиотеки, за счёт  
этого, увеличен.

# Динамическая библиотека

## Linux, GCC

**ldd** - утилита Linux, вывод  
динамических зависимостей

Все зависимости, включая пути,  
откуда они будут загружены. И наша  
библиотека - тоже.

```
1. > ldd dynamic_library_usage
2.      linux-vdso.so.1 (0x00007ffc4b6f6000)
3.      libsumLib.so => /home/cloudshell-user/webinar-libraries/dynamic_lib/build/lib/libsumLib.so
4.      libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f8484f16000)
5.      libm.so.6 => /lib64/libm.so.6 (0x00007f8484bd6000)
6.      libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f84849c0000)
7.      libc.so.6 => /lib64/libc.so.6 (0x00007f8484615000)
8.      /lib64/ld-linux-x86-64.so.2 (0x00007f848549a000)
```

# Динамическая библиотека

Windows, MS Visual C++

CMakeLists.txt

см. "dynamic\_lib/lib" в  
приложенном примере

Просто поменять  
"STATIC" → "SHARED"  
не достаточно!

1

sum.h

```
#pragma once

namespace library {

    int sum(int a, int b);

} // namespace library
```

3

```
cmake_minimum_required(VERSION 3.5)
project(dynamic_library)

add_library(dynamicLib SHARED
    sum.h
    sum.cpp
)
```

sum.cpp

2

```
#include "sum.h"

namespace library {

    int sum(const int a, const int b) {
        return a + b;
    }

} // namespace library
```

4

LINK : fatal error LNK1104: cannot open  
file 'lib\Debug\sumLib.lib'  
[dynamic\_lib\build\dynamic\_library\_usage.v  
cxproj]

# Динамическая библиотека

Windows, MS Visual C++

- Вводится понятие **видимости символов**
- ... или экспорта символов
- Только экспортированные символы можно использовать
- Поэтому создаётся таблица экспорта
- Которая представляет собой файл .lib
- Но это **не статическая библиотека** - это просто таблица экспорта
- Нет явно помеченных символов для экспорта - **таблица экспорта не создаётся**

# Динамическая библиотека

## Windows, MS Visual C++

sum.h для экспорта

```
#pragma once

namespace library {

    __declspec(dllexport) int sum(int a, int b);

} // namespace library
```

# Динамическая библиотека

Windows, MS Visual C++

sum.h для экспорта

```
#pragma once

namespace library {

    __declspec(dllexport) int sum(int a, int b);

} // namespace library
```

sum.h для импорта

```
#pragma once

namespace library {

    __declspec(dllimport) int sum(int a, int b);

} // namespace library
```

# Динамическая библиотека

Windows, MS Visual C++

sum.h для экспорта

```
#pragma once

namespace library {

    __declspec(dllexport) int sum(int a, int b);

} // namespace library
```

sum.h для импорта

```
#pragma once

namespace library {

    __declspec(dllimport) int sum(int a, int b);

} // namespace library
```

Заголовок один, а деклараций - **две**.



# Динамическая библиотека

## Windows, MS Visual C++

sum.h - один за всех

```
#pragma once

#if defined(_WIN32)
#   ifdef BUILD_SUM_DLL
#       define SUM_DLL_API __declspec(dllexport)
#   else
#       define SUM_DLL_API __declspec(dllimport)
#   endif
#else
#       define SUM_DLL_API
#endif

namespace library {

    SUM_DLL_API int sum(int a, int b);

} // namespace library
```

# Динамическая библиотека

Windows, MS Visual C++

sum.h - один за всех

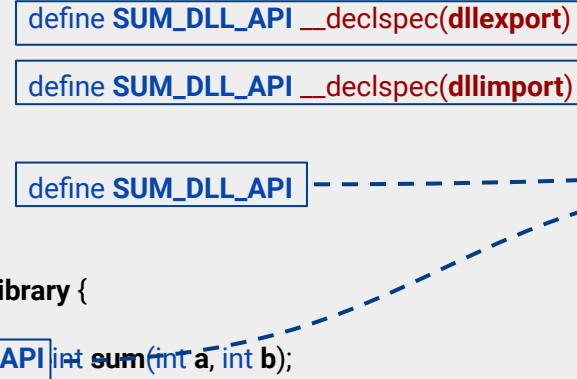
```
#pragma once

#if defined(_WIN32)
#   ifdef BUILD_SUM_DLL
#       define SUM_DLL_API __declspec(dllexport)
#   else
#       define SUM_DLL_API __declspec(dllimport)
#   endif
#else
#   define SUM_DLL_API
#endif

namespace library {

    SUM_DLL_API int sum(int a, int b);

} // namespace library
```



Условная компиляция:

1. Это Windows?
2. Это экспорт?
3. Это импорт?

# Динамическая библиотека

Windows, MS Visual C++

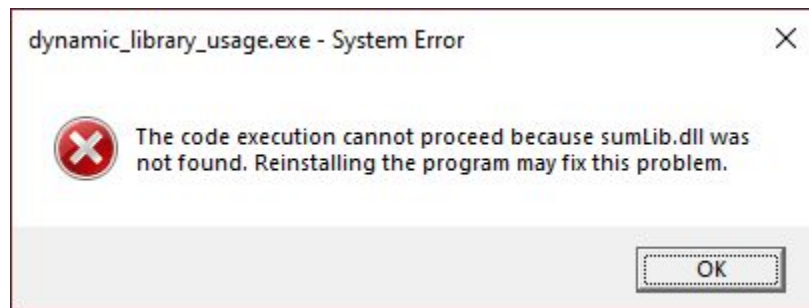
Утилита пакета VS **DUMPBIN**.  
Команда **/DEPENDENTS** - вывод  
динамических зависимостей.

Все зависимости.  
И наша библиотека - тоже.

```
01. > dumpbin /DEPENDENTS dynamic_library_usage.exe
02.
03. File Type: EXECUTABLE IMAGE
04.
05. Image has the following dependencies:
06.
07.    sumLib.dll
08.    MSVCP140D.dll
09.    VCRUNTIME140_1D.dll
10.    VCRUNTIME140D.dll
11.    ucrtbased.dll
12.    KERNEL32.dll
```

# Динамическая библиотека

Windows, MS Visual C++



Ошибка поиска DLL при старте не всегда наглядна.

А если это LoadLibrary - вообще зависит от того, как программист на неё реагирует.

# Когда что применять?

## Статические библиотеки:

- по умолчанию

## Динамические библиотеки:

- для плагинов
- для проектов с зависимостями
- при необходимости быстрой замены реализации
- при необходимости динамической загрузки и выгрузки



# Следующий вебинар



20 июня 2023

## Современные средства автоматизации. Пакетные менеджеры, снова CMake



Ссылка на вебинар  
будет в ЛК за 15 минут



Материалы  
к занятию в ЛК —  
можно изучать



# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет



**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**