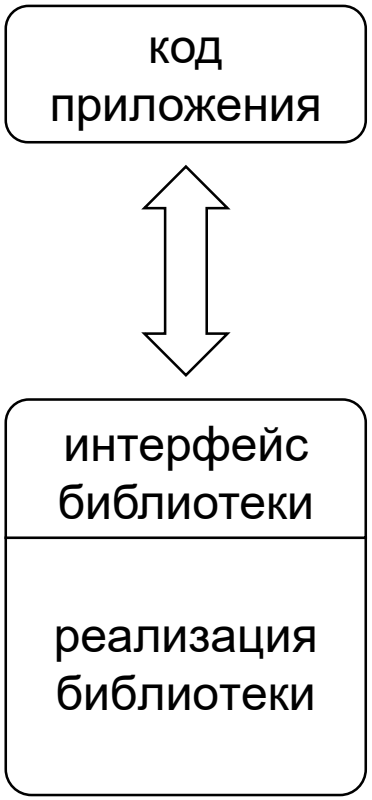
Использование внешних программных библиотек

«Технология программирования»
осенний семестр 2016 г.

Библиотеки

- Решения многих сложных задач (разбор форматов данных, обработка данных) уже есть в виде библиотек, и могут быть использованы в других программах.
- Физически, библиотека — это готовый код:
  - исходный код;
  - объектный код (статические библиотеки);
  - двоичный код (динамические библиотеки).
- Задачи:
  - Выбор библиотеки, изучение её интерфейса.
  - Поиск и установка библиотеки.
  - Подключение библиотеки к приложению.
  - (Удобное, безопасное) использование библиотеки из приложения.

Интерфейс библиотеки



- **Application Programming Interface (API) —** интерфейс программирования приложений: функции и типы данных, с помощью которых код приложения взаимодействует с библиотекой.
- API документируется автором библиотеки.
- Обычно предоставляется заголовочный файл, где описано API.

Интерфейс «в стиле C»

- Есть у большинства библиотек, даже написанных не на C.
- Средства C — «наибольший общий делитель» для многих языков:
  - целые и вещественные типы данных, **void**;
  - структуры без методов;
  - обычные функции;
  - указатели (не ссылки).
- Над таким интерфейсом может быть надстройка (binding), которая оборачивает (wraps) вызовы в более удобные конструкции.
  - Особенно актуально для интерпретируемых языков, из которых можно, но неудобно использовать интерфейс в стиле C.

Виды библиотек по способу связывания

- **Header-only:**
  один или несколько заголовочных файлов.
  - Boost.Any, Boost.Variant, …, utf8pp
  - Выполняется на стадии компиляции.
- **Статически компонуемые (static):**
  специальный архив с объектными файлами (*.a, *.lib).
- **Динамически подключаемые (dynamic):**
  исполняемый код (*.so, *.dll), загружаемый после запуска использующей программы (иногда еще небольшая статическая библиотека, которая это делает).

Включение кода библиотеки в состав проекта

Небольшие библиотеки может быть удобнее включить в состав проекта наряду с другими файлами.

- Удобно вносить доработки в библиотеку.
  - Лицензия важна!
- Иногда технически сложно:
  - система сборки библиотеки иная, чем у проекта;
  - библиотека не на языке проекта.
- Иногда невозможно:
  - закрытый исходный код;
  - лицензионные ограничения.
    - Если библиотека под GPL, при включении её кода в проект нужно открывать код всего проекта.

Статическое и динамическое связывание

Static

- Отсутствие у программы внешних зависимостей.
- Невозможность обновления библиотеки отдельно от программы.
- Увеличение размера файла.
- Иногда ограничено лицензией.
  - Пример: LGPL.
- Иногда технически невозможно.

Dynamic

- Вместе с программой в системе нужны библиотеки.
- Обновить библиотеку можно без пересборки программы.
- Файл программы компактен

  (выгодно, если многие программы используют одни и те же библиотеки).

Установка библиотеки

- Linux, *BSD: из пакетного менеджера.
  - Устанавливаются в стандартные каталоги.
  - Как правило, поставляются с документацией.
- Windows: загрузка с официального сайта.
  - Обычно в виде архива с двоичными файлами или кодом.
- OS X (macOS): Homebrew или как в Windows.
- Может быть доступно несколько версий:
  - по разрядностям (32 бита — x86 или 32bit, 64 бита — x64 или x86_64);
  - для разных операционных систем;
  - для компиляторов, например:
    - GNU C++ Compiler (GCC),
    - Microsoft Visual C++ (MSVC)
      - с разными вариантами стандартной библиотеки (MT, ST);
  - с отладочной информацией (суффикс d или –dbg).

CMake спешит на помощь!

- Может находить библиотеку, пути к её файлам.
- Требуется, чтобы авторы библиотеки написали модуль расширения для CMake.
  - Для многих библиотек поставляется с CMake (например).
- Пример:

```
# Найти cURL. Если не найдена, сборка невозможна.
find_package(curl REQUIRED)

# Добавить путь к заголовочным файлам в стандартные.
include_directories(${CURL_INCLUDE_DIRS})

# Компоновать проект с библиотекой.
target_link_libraries(
    ${PROJECT_NAME}, ${CURL_LIBRARIES})
```

cURL: передача данных по сети

- https://curl.haxx.se/libcurl/
- «Курл» (в России), kurl [kö̈l] (авторы),
  see URL «си Ю-эР-эЛ» (многие).
- Позволяет загружать web-страницы, файлы, почту,
  отправлять запросы по 23 протоколам.
- Позволяет настроить очень многое,
  но также и загрузить web-страницу в 5 строк.
- На основе cURL написана одноименная программа.

Пример: загрузка web-страницы

```
// Инициализация сеанса связи (представленного как CURL*).
CURL *curl = curl_easy_init();

if (curl) {

    // Указание библиотеке адреса страницы для загрузки.
    curl_easy_setopt(curl, CURLOPT_URL, "http://example.com");

    // Загрузка страницы (печатается на экран).
    CURLcode result = curl_easy_perform(curl);

    // Освобождение ресурсов, использованных библиотекой.
    curl_easy_cleanup(curl);

}
```

А что такое тип CURL?

А как выполнить другие действия?

А зачем?

Opaque pointer (handle)

- По-русски: дескриптор, *(редко)* описатель.
- Идентификатор объекта библиотеки,
  устройство которого от пользователя скрыто.
  - Часто указатель на структуру, которая объявлена,
    но не определена в API (пример: curl, http_parser).
  - Иногда тип-синоним целого числа (пример: OpenGL).
- По сути, инкапсуляция средствами C.
  - Да, ООП — это парадигма, язык не обязан иметь классы,
    чтобы использовать в нем ООП.
- Операции над дескриптором:
  - сохранить в переменную;
  - передать в функцию библиотеки.

«Тонкие обертки» (thin wrappers)

- Классы и функции, основная задача которых — вызвать библиотечные функции с минимальной дополнительной логикой.
  - Например, можно помимо вызова библиотечной функции проверять результат и преобразовывать его в формат приложения.
  - Пример далее — тонкая обертка.
- Скрывается (изолируется) использование конкретной библиотеки.
- Выделяется интерфейс подсистемы, которую реализует библиотека.
- Становится возможным заменить библиотеку, не меняя остальную программу.

Ресурсы, их утечка и борьба с ней

- **[R]esource** is any physical or virtual component of limited availability within a computer system *(«Википедия»).*
  - Ресурс есть то, чего может не хватать.
- Примеры ресурсов: оперативная память, место на диске, пропускная способность сети.
- Выделенные ресурсы необходимо освобождать, иначе случится их утечка (исчерпание со временем).
  - Пример: утечка памяти, если использовать **new** без **delete.**
- Проблема: можно забыть освободить ресурсы.
- Компилятор никогда не забывает освободить память под переменные. **Можно ли использовать это для ресурсов?**

Что нужно (ideal)?

```
{
    Session session = curl_easy_init();
    curl_easy_setopt(session, CURLOPT_URL, "http://example.com");
    curl_easy_perform(session);
}
```

Здесь уничтожается Session и должна же быть вызвана curl_easy_cleanup().

В перспективе («толстая» обертка):

```
Session session;
session.download("http://example.com");
```

Обертка для cURL (1)

```
class Session {
public:
    Session(CURL* curl) {
        curl_ = curl;
    }
    ~Session() {
        if (curl_) {
            curl_easy_cleanup(curl_);
        }
    }
private:
    CURL* curl_;
};
```

конструктор

деструктор

При создании (инициализации) Session ресурс захватывается.

При уничтожении Session ресурс освобождается.

RAII

Кроме объекта Session никто не может повредить (заменить) дескриптор.

инкапсуляция

Объектно-ориентированное программирование (кратко)

- Технически всё просто:
  - Класс = данные (поля) + функции (методы).
  - Класс похож на структуру и набор функций над ней:

```
class Example {
public:
    int field;
    void method() {
        cout << field << '\n';
    }
};
```

```
struct Example {
    int field;
};
void Example_method(Example* this) {
    cout << this -> field << '\n';
}
```

- Идея ООП: программирование — это моделирование.
- Session — модель сеанса, т. е. упрощенное представление:
  - Скрывает механизм работы с cURL.
  - Оставляет понятные операции: создать и завершить сеанс (пока).

Инкапсуляция

- Объект управляет своим состоянием, оно скрыто от внешнего кода и не может быть испорчено им.
- В C++ реализуется через уровни доступа (**public**-члены доступны извне, **private** — нет).
  - Есть еще **protected**, будет рассмотрено позже.

Разница между **struct** и **class** — только уровень доступа по умолчанию:

```
struct X
{
    // …
```

⟷

```
class X
{
public:
    // …
```

```
struct X
{
private:
    // …
```

⟷

```
class X
{
    // …
```

Обертка для cURL (2)

- Проблема: Session — не CURL*, не работает код
  ```
  curl_easy_setopt(session, …);
  ```
- Решение: дать способ преобразовать Session в CURL*:

```
class Session {
    // …
    operator CURL*() {
        return curl_;
    }
};
```

Оператор преобразования Session в CURL*.
Вызывается в случаях, когда нужен CURL*,
а передано Session.

Тип возвращаемого значения не указывается,
т. к. для оператора преобразования типа вариантов нет.

Владение ресурсом

- Владелец (owner) ресурса отвечает за освобождение этого ресурса.
  - Session включает CURL*, который освобождает в деструкторе.
- Иногда владелец также и выделяет ресурс.
  - Можно вызывать curl_easy_init() в конструкторе Session.
- Не должно быть конфликтов владения.

```
Session s1{curl_easy_init()};
Session s2 = s2;
// При освобождении второго объекта
// будет попытка освободить ресурс второй раз — ошибка.
```

  - Нужно запретить копирование Session:

```
Session(const Session&) = delete;
Session& operator=(const Session&) = delete;
```

Функции обратного вызова

- По-английски «callback», по-русски также «обработчик».
- Библиотеке передается указатель на функцию, которая будет вызвана библиотекой при некотором событии.
  - Аргументы функции будут содержать информацию о событии. Пример: событие — получение данных, аргументы — данные и их размер.
  - Функция должна иметь определенную сигнатуру (количество, порядок и типы аргументов, тип возвращаемого значения).

Пример callback для cURL

Данные загружаются из сети порциями, on_data_received() будет вызвана при загрузке очередной порции.

```
size_t on_data_received(char* data, size_t block_count,
        size_t block_size, void* userdata) {
    const size_t byte_count = block_count * block_size;
    cout << "Received " << byte_count << " bytes:\n";
    cout.write(data, byte_count);
    return byte_count;
}
// …

curl_easy_setopt(
        curl, CURLOPT_WRITEFUNCTION, on_data_received);
```

Что такое userdata?

- «Пользовательские данные», или «контекст».
- Произвольные (заданные пользователем) данные, которые будет получать callback при каждом вызове.
- Это типовой прием.
- Userdata указывается при установке callback или отдельно, см. документацию.
  - В cURL для CURL_WRITEFUNCTION — отдельно.

Использование userdata в cURL

Перед установкой callback выбирается, в какой файл записывать данные, а callback узнает это через userdata.

```
size_t on_receive(char* data, size_t blocks, size_t count,
        void* userdata) {
    auto handle = reinterpret_cast<FILE*>(userdata);
    fwrite(data, blocks, size, handle);
    return blocks * size;
}

// …

FILE* handle = fopen("downloaded.file", "w");
curl_easy_setopt(curl, CURL_WRITEFUNCTION, on_receive);
curl_easy_setopt(curl, CURL_WRITEDATA, handle);
curl_easy_perform(curl);
fclose(handle);
```

Callback, userdata и ООП

- Часто нужно, чтобы в качестве callback вызывался метод того объекта, который callback поставил (или еще какого-то).
- Однако:
  - Callback не может быть методом объекта*.
    - Не хватает информации: для вызова метода нужен объект, а callback определяет только функцию.
  - Библиотека не отслеживает, откуда установлен callback.
- Решение:
  - Одна функция-диспетчер для всех объектов.
  - В userdata передавать адрес объекта,
    и у него вызвать из диспетчера «настоящий» обработчик.

* См. также std::function, std::bind(), boost::bind(), но для многих библиотек (в т. ч. cURL) это не подходит, и выходит за рамки лекции.

Диспетчеризация callback’ов (1)

```
using Data = void*;       // Условные полезные данные.

// «Слушатель событий» (типовое название) со своим именем.
class Listener {
public:
    Listener(const std::string& name) : name_{name} {}

    void on_event(Data data) {
        cout << "Listener " << name_ << " handles event "
             << "with data=" << data << "\n";
    }

private:
    std::string name_;
};
```

Диспетчеризация callback’ов (2)

- Функция-диспетчер извлекает из userdata указатель на объект и передает данные его обработчику:

```
void dispatch(void* data, void* userdata) {
    auto listener = reinterpret_cast<Listener*>(userdata);
    listener->on_event(data);
}
```

- Listener a("A"), b("B");
- При установке callback адреса объектов передаются как userdata (условный API):

```
set_callback_with_userdata(…, &dispatch, &a);
set_callback_with_userdata(…, &dispatch, &b);
```

- Далее dispatch() может вызываться для разных объектов в неизвестном заранее порядке.

Диспетчеризация callback'ов (3)

- Объекты (в примере — a и b) не должны уничтожаться, пока может быть вызвана функция-диспетчер. Неправильно:

```
if (…) {
    Listener a("A");
    set_callback_with_userdata(…, &dispatch, &a);
}  // Здесь a уничтожается, обращаться по её адресу будет нельзя!
do_something_that_may_trigger_callback();
```

  Вариант решения — разместить a в куче (new Listener("A")).

- Вместо глобальной функции dispatch() лучше использовать статический метод, а on_event() сделать **private:**

```
class Listener {  // …
public:
    static void dispatch(void* data, void* userdata);
};
// …
set_callback_with_userdata(…, &Listener::dispatch, &a);
```

Лицензирование библиотек

- Лицензия обычно распространяются вместе с библиотекой.
  - Файл LICENSE или подобный.
  - И дублируется в исходном коде в комментариях.
- В зависимости от способа подключения, лицензия библиотеки может влиять на лицензию всей программы.
- Большинство лицензий требуют, по крайней мере, упоминать о библиотеке в лицензии конечного продукта.
- https://tldrlegal.com/ — объяснения типовых лицензий простым языком.
  - Квалифицированное заключение может дать только юрист!