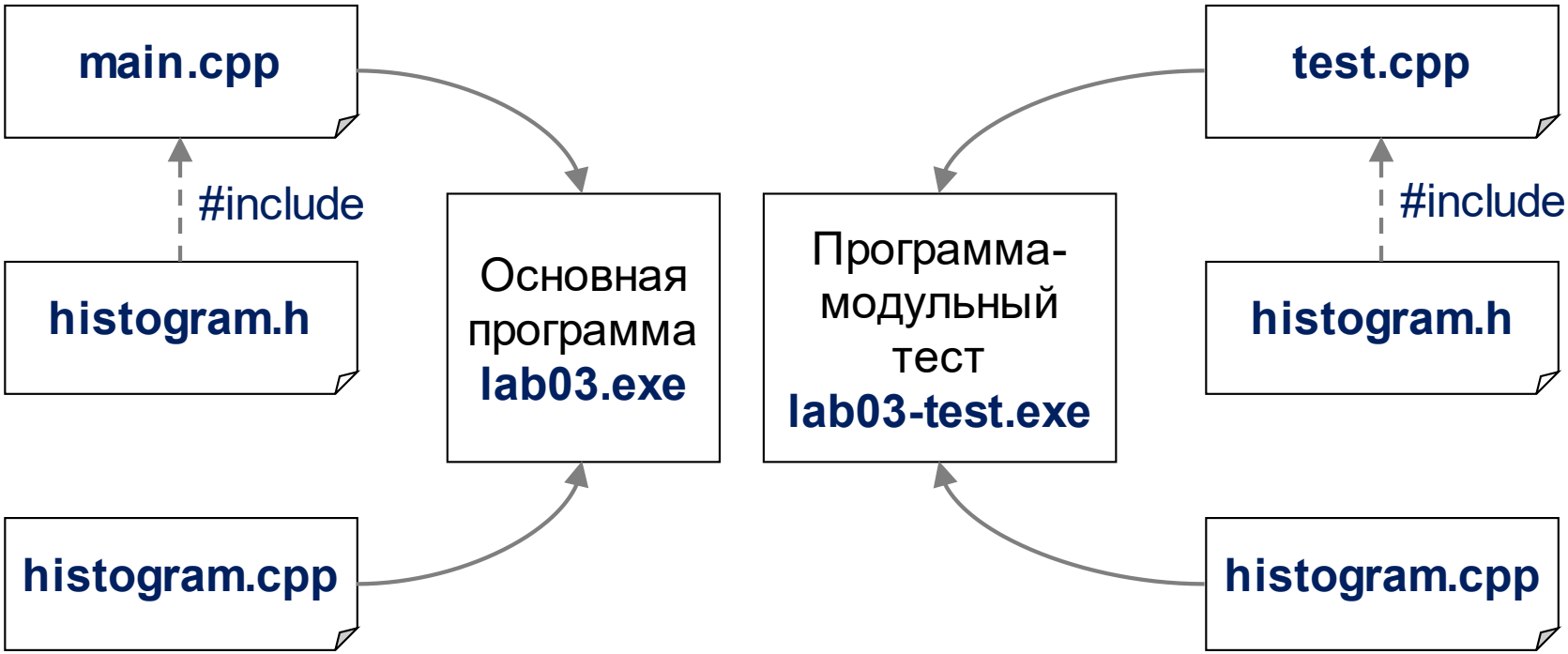
Библиотеки

Разработка ПО систем управления

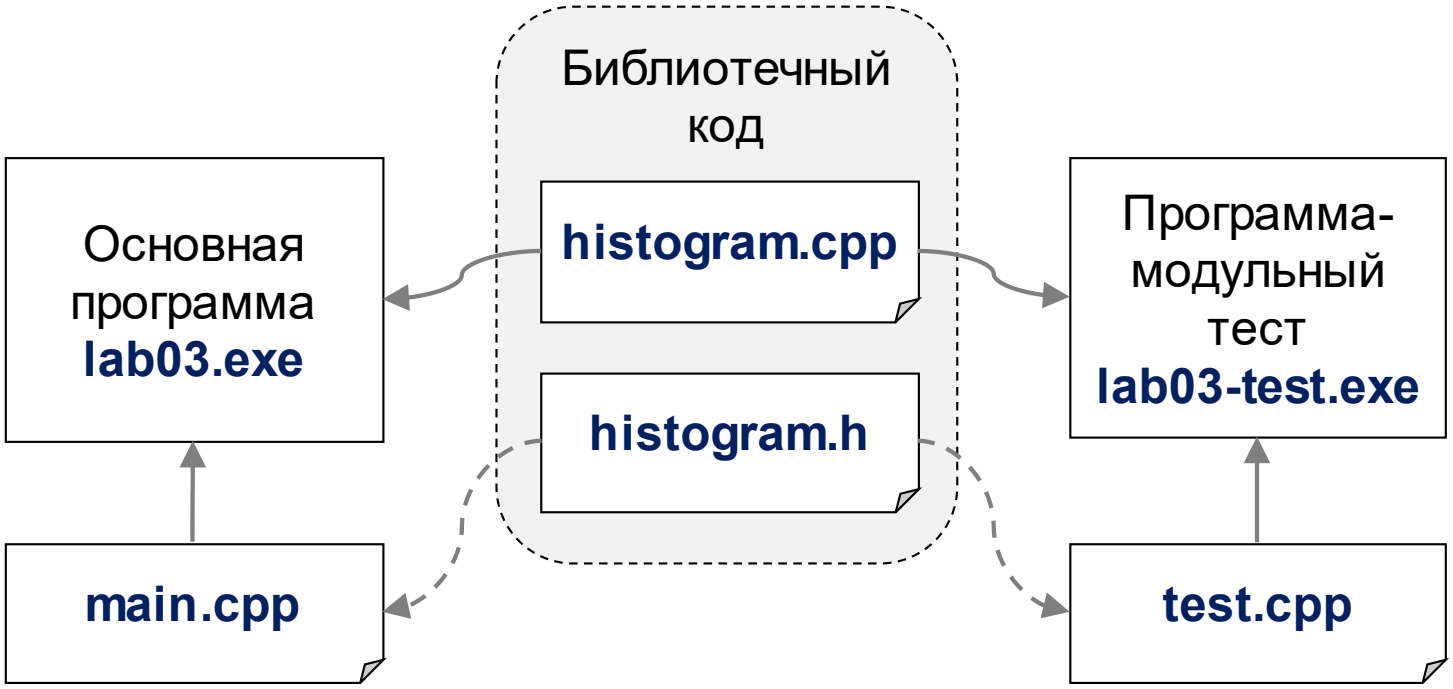
Кафедра Управления и интеллектуальных технологий

Весна 2021

Пример: ЛР № 3, общий код



Пример: ЛР № 3

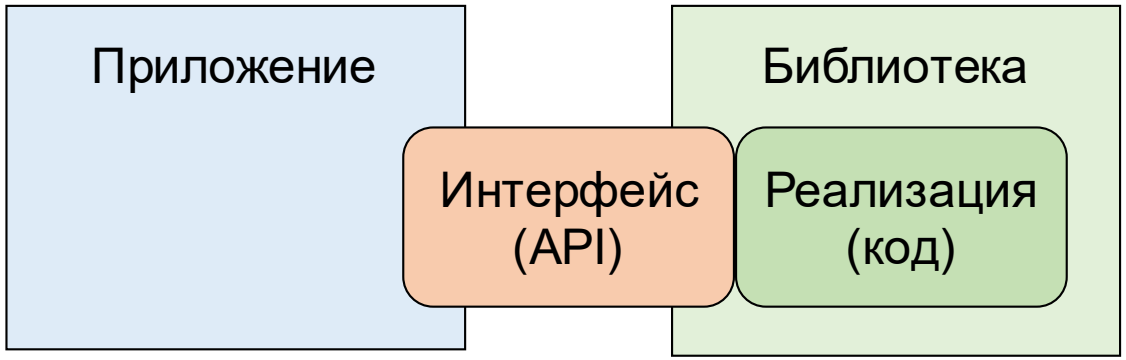

Библиотечный код
histogram.cpp
histogram.h
Основная программа lab03.exe
main.cpp
Программа-модульный тест lab03-test.exe
test.cpp

Почему не включать код библиотек в программу?

- Библиотека может быть на другом языке.
- Код библиотеки может быть сложно собрать.
  - Другой компилятор (MSVC, LLVM, GCC, …)
  - Другая сборочная система (Make, MSBuild, ...)
  - Зависит от других библиотек (транзитивно).
- Код библиотеки может быть закрыт.
  - Коммерческие: Intel MKL (расчеты), графические движки.
  - Системные библиотеки Windows.

Терминология

- **Приложение** (application) — программа, которая использует библиотеку.
- **Интерфейс** — средства взаимодействия приложения с библиотекой (функции и типы).
  - **API** = application programming interface.
  - Для C и C++ интерфейс описан в файлах *.h.



Выбор библиотеки

- Функциональность
- Лицензия
  - Цена
  - Условия использования
- Наличие под нужные платформы (ОС)
- Удобство API
  - Формат библиотеки (header-only, статическая, DLL)
  - C-style API
- Документация

Лицензии

TLDRLegal - Software Licenses Explained in Plain English

- Проприетарные
  - Обычно нельзя применять вне компании и без согласия **правообладателя** (не автора!).
  - Copyright © 2021 MPEI. All rights reserved.
- Необременительные
  - Использование (почти) не создает обязательств.
  - MIT, BSD, Apache, …
- Обременительные («вирусные»)
  - Накладывает ограничения на лицензию **программы.**
  - GPL, LGPL, …

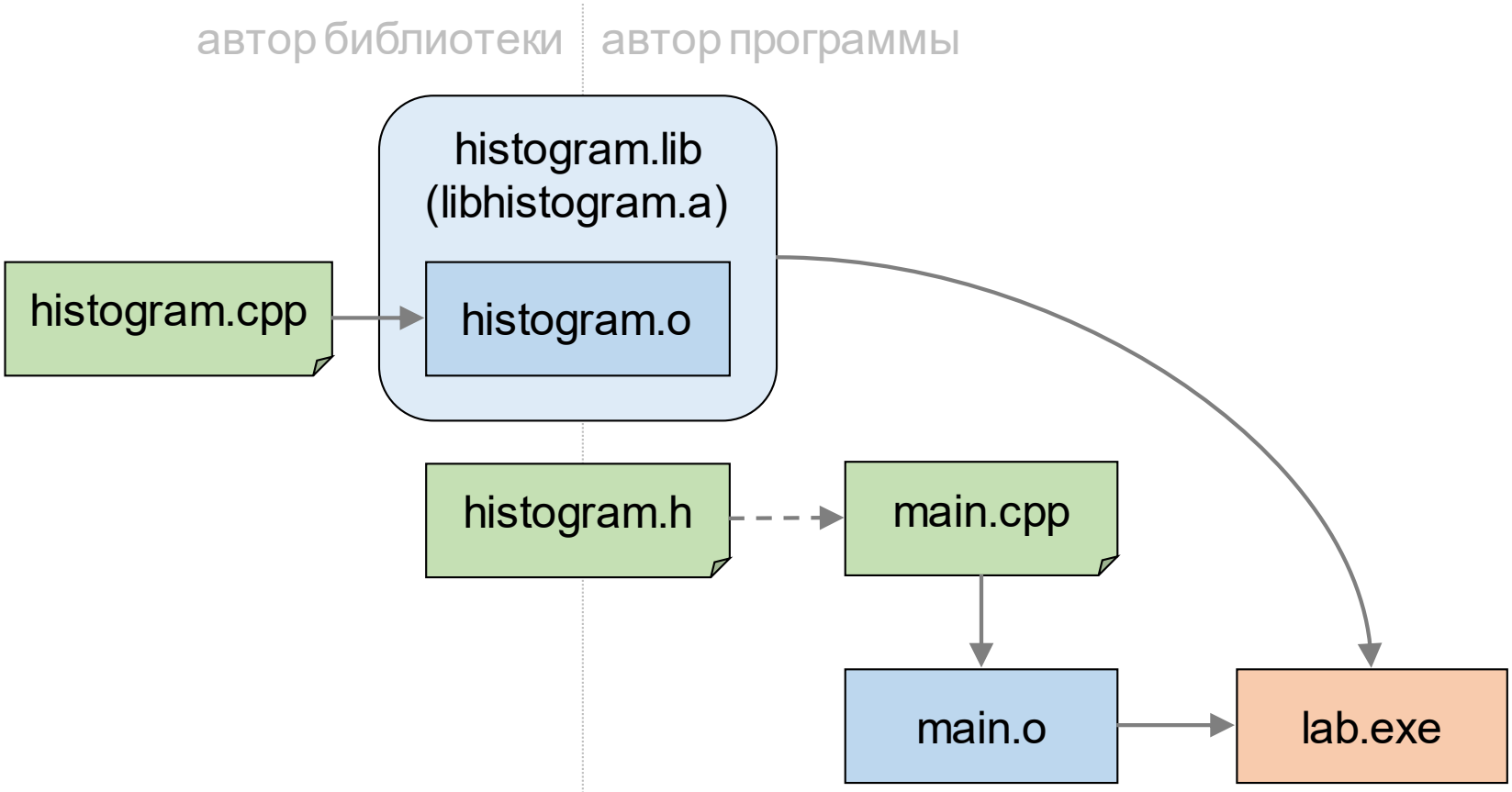
Header-only библиотеки

- Подключение — просто #include.
- Могут использовать все возможности языка.
- Лучше оптимизируются.

- Должны быть на языке программы (C или C++).
- Растет время компиляции
  (библиотека компилируется для каждого использующего файла).

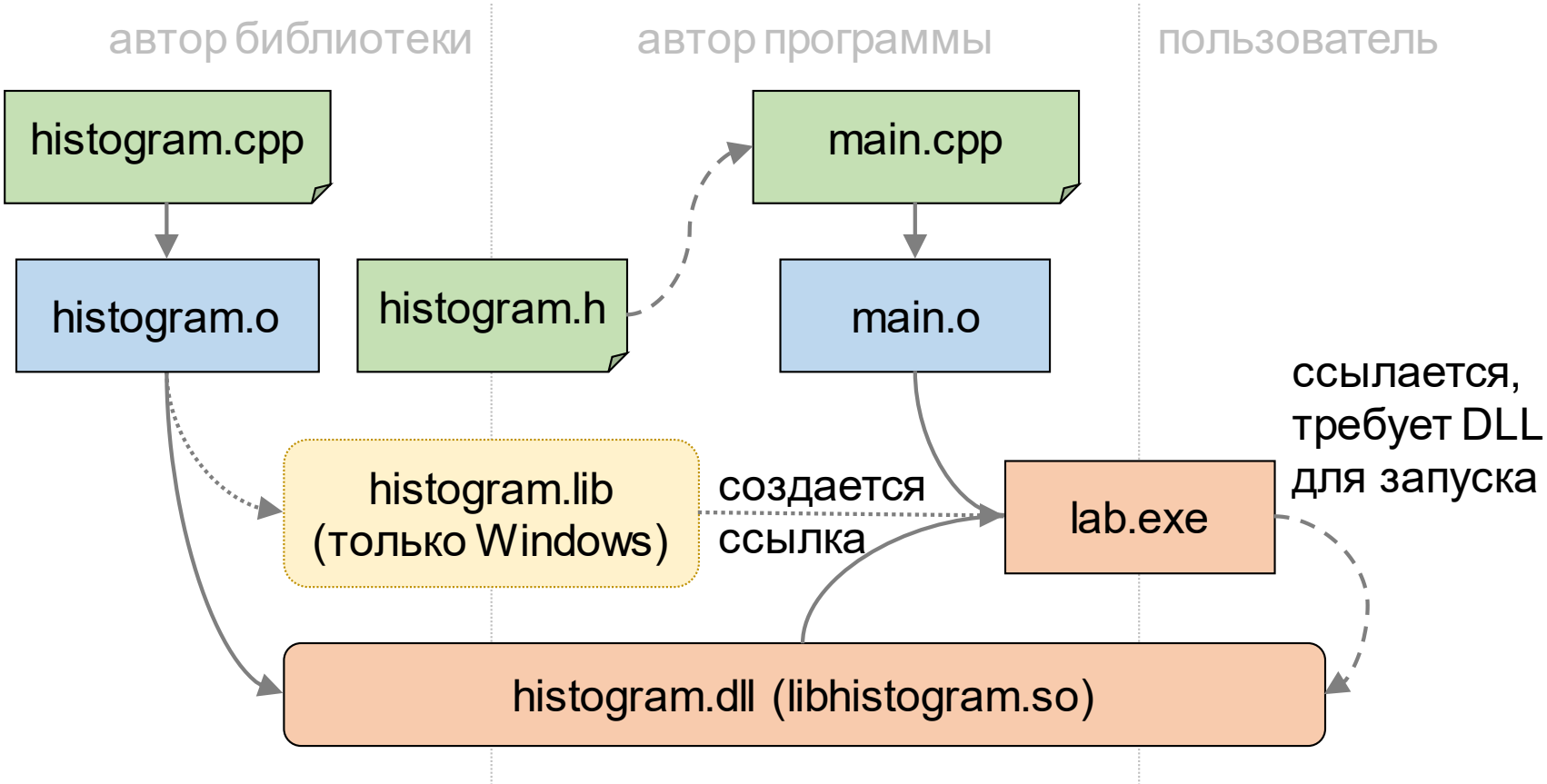
Статические библиотеки

архив с объектным кодом + заголовочные файлы



Динамические библиотеки

заголовочный файл + файл с машинным кодом (DLL)



Статические библиотеки

+ Программа запускается без дополнительных файлов DLL.
+ Работа программы не зависит от установленных у пользователя библиотек.
− Чтобы обновить библиотеку, нужно пересобирать программу. *
− Больший размер файла.

Динамические библиотеки

+ Можно обновить библиотеку без пересборки программы. *
+ Обновление библиотеки исправляет все программы, использующие её.
+ Меньший размер файла.
− EXE не запустится без библиотек.
− Если версии установленных библиотек отличаются, EXE может работать неверно.

* Решение — bundling: использовать DLL, но устанавливать нужные их версии в каталог с программой.

Пример программы, использующей DLL:

```
dir origin.exe
   ...
   1 File(s)      3 137 808 bytes
dir *.dll
  ...
  42 File(s)    165 323 632 bytes
```

Гораздо меньшая программа,
использующая примерно тот же набор библиотек (Qt5),
но собранная статически, имеет EXE размером **~17 МБ.**

Когда весь код в одном EXE, это может быть лучше
для производительности.

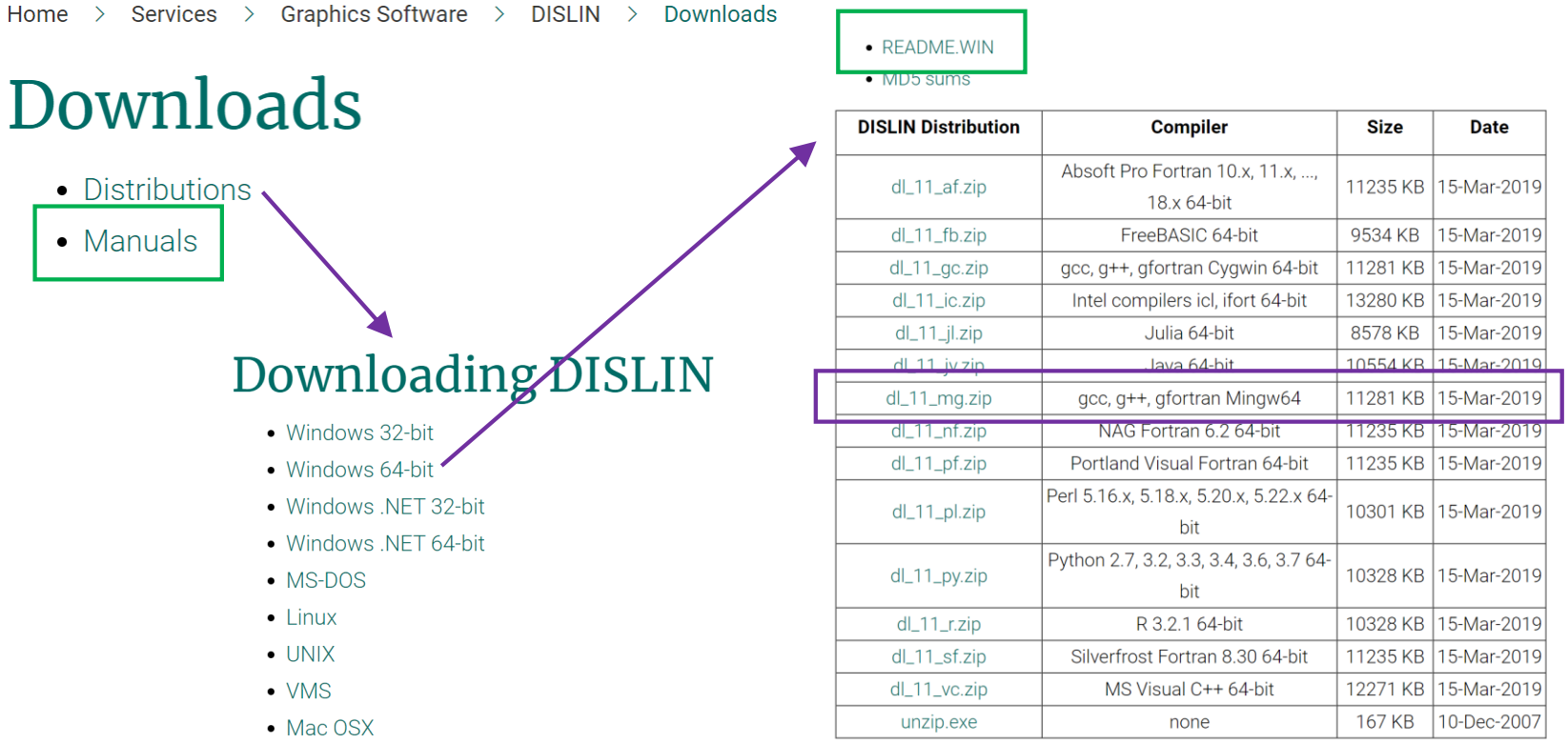
Подключение библиотек

1. Получить файлы библиотеки (*.h, *.lib, *.dll, …).
   - Загрузить или скопировать вручную.
   - Установить в систему штатными средствами.
   - Установить через менеджер библиотек.
2. Компилятор и компоновщик должны их найти.
   - Разместить в стандартных каталогах (папках).
   - Настроить каталоги для поиска.
3. Программа при запуске должна найти DLL.
   - Устанавливать библиотеку вместе с программой.
   - Разместить DLL в каталоге, где происходит поиск.

Загрузка библиотеки вручную

[Полный пример использования DISLIN в учебной практике.]

Библиотеку нужно брать из доверенных источников:
с сайта разработчика, по ссылке с него или из репозитария пакетов.



DISLIN Distribution	Compiler	Size	Date
dl_11_af.zip	Absoft Pro Fortran 10.x, 11.x, ..., 18.x 64-bit	11235 KB	15-Mar-2019
dl_11_fb.zip	FreeBASIC 64-bit	9534 KB	15-Mar-2019
dl_11_gc.zip	gcc, g++, gfortran Cygwin 64-bit	11281 KB	15-Mar-2019
dl_11_ic.zip	Intel compilers icl, ifort 64-bit	13280 KB	15-Mar-2019
dl_11_jl.zip	Julia 64-bit	8578 KB	15-Mar-2019
dl_11_jv.zip	Java 64-bit	10554 KB	15-Mar-2019
dl_11_mg.zip	gcc, g++, gfortran Mingw64	11281 KB	15-Mar-2019
dl_11_nf.zip	NAG Fortran 6.2 64-bit	11235 KB	15-Mar-2019
dl_11_pf.zip	Portland Visual Fortran 64-bit	11235 KB	15-Mar-2019
dl_11_pl.zip	Perl 5.16.x, 5.18.x, 5.20.x, 5.22.x 64-bit	10301 KB	15-Mar-2019
dl_11_py.zip	Python 2.7, 3.2, 3.3, 3.4, 3.6, 3.7 64-bit	10328 KB	15-Mar-2019
dl_11_r.zip	R 3.2.1 64-bit	10328 KB	15-Mar-2019
dl_11_sf.zip	Silverfrost Fortran 8.30 64-bit	11235 KB	15-Mar-2019
dl_11_vc.zip	MS Visual C++ 64-bit	12271 KB	15-Mar-2019
unzip.exe	none	167 KB	10-Dec-2007

C-style API

- «Простые» типы данных: uint32_t, char, double.
  - Присутствуют во всех языках.
  - Имеют известное представление в памяти.
  - Типы фиксированного размера: uint32_t, ...
- Структуры из полей простых типов.
- Статические массивы.
- Указатели, в т. ч. для динамических массивов.
- Функции.

НЕТ: типы со сложным внутренним устройством, которые специфичны для языка (vector, string).

Пример: библиотека для работы с матрицами

```
struct Matrix {
        uint32_t rows;
        uint32_t cols;
        double* items;
};
uint32_t matrix_create(Matrix* m, uint32_t rows, uint32_t cols);
void matrix_free(Matrix* m);
double matrix_get(const Matrix* m, uint32_t row, uint32_t col);
void matrix_set(Matrix* m, uint32_t row, uint32_t col,
        double value);
void matrix_multiply(const Matrix* a, const Matrix* b, Matrix* c);
```

Проектирование API начинается с примеров использования

```
Matrix a, b, c;
if (matrix_create(&a, 3, 4) < 0) {
        cerr << "Не хватает памяти под A.\n";
        return 1;
}
for (uint32_t i = 0; i < 3; i++) {
        for (uint32_t j = 0; j < 4; j++) {
                double x;
                cin >> x;
                matrix_set(&a, i, j, x);
        }
}
// Аналогичный код для ввода b.
matrix_multiply(&a, &b, &c); // c = a * b
```

Безопасный API

- Библиотека должна сообщать об ошибках.
  - Функция может возвращать признак успеха (обычно 0).
  - Может быть функция получения последней ошибки.
- Библиотека НЕ должна реагировать на ошибки:
  - НЕ показывать сообщения (не знает, как нужно).
  - НЕ завершать программу аварийно (не знает, стоит ли).
- Вызовы функций или выполняют работу,
  или не меняют ничего
  (НЕ делают работу наполовину).

Безопасный API создания матриц

```
uint32_t matrix_create(Matrix* m, uint32_t rows, uint32_t cols)
{
    double* items = (double*)calloc(rows * cols, sizeof double);
    if (items == nullptr) {
        return -1;
    }
    m->items = items;
    m->rows = rows;
    m->cols = cols;
    return 0;
}
```

Выделяет память под rows*cols элементов типа **double**.
Возвращает **nullptr** при ошибке.

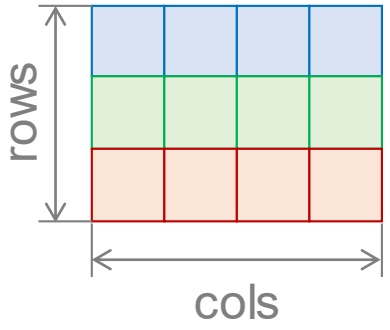
Сообщается об ошибке.
Если приложению будет нужно,
оно само покажет ошибку пользователю.

Результат записывается только после того, как проверены все ошибки.
Если была ошибка, m не меняется.

0 значит «успех».
Это документируется.

Доступ к элементам матрицы

Программе не должно быть нужно работать с «внутренностями» структур библиотеки.




```
struct Matrix {
    uint32_t rows;
    uint32_t cols;
    double* items;
};
```

```
double matrix_get(const Matrix* m,
    uint32_t row, uint32_t col)
{
    if ((row >= m->rows) || (col >= m->cols)) return 0;
    return m->items[row * m->cols + col];
}
```

Безопасный API:
библиотека
не обращается
за пределы массива
при ошибочных
аргументах функции.
Но это «лишние»
операции.

Opaque pointers (handles)

— Зачем программе знать содержимое структуры Matrix?

— Незачем!

Хуже того: автор библиотеки теперь не может легко изменить состав полей.

matrix.h (интерфейс)

```
struct Matrix;

Matrix* matrix_create(
        uint32_t rows, uint32_t cols);
```

matrix.cpp (реализация)

```
struct Matrix {
        uint32_t rows;
        uint32_t cols;
        double* items;
};
```

Можно объявлять (но не разыменовывать) **непрозрачные** указатели на объявленные, но не определенные структуры.

Handle (дескриптор) — объект библиотеки, устройство которого скрыто.

Opaque pointers [продолж.]

```
Matrix* matrix_create(uint32_t rows, uint32_t cols)
{
        Matrix* m = (Matrix*)malloc(sizeof(*m));
        if (m == nullptr) {
                return nullptr;
        }
        m->data = (double*)calloc(rows * cols, sizeof(double));
        if (m->data == nullptr) {
                free(m);
                return nullptr;
        }
        m->rows = rows;
        m->cols = cols;
        return m;
}
```

Бибиотека выделяет память и под структуру, и под элементы матрицы.

Если этого не сделать, память под Matrix «утечет» (memory leak).

Библиотеке доступно определение **struct** Matrix { … }, её функции могут обращаться к полям структуры.

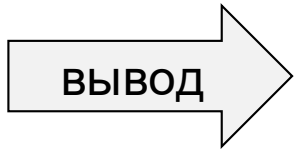
Указатели на функции

// Выполнить *операцию* над каждым элементом.

void matrix_op(**const** Matrix* m, ***операция***);

```
void print(Matrix* m, uint32_t i, uint32_t j) {
    cout << "m[" << i << ", " << j << "] = " << matrix_get(m, i, j) << "\n";
}

int main() {
    auto a = matrix_create(2, 3);
    // Заполнение матрицы a = [1 2 3; 4 5 6]
    matrix_op(a, print);
}
```




```
m[0, 0] = 1
m[0, 1] = 2
m[0, 2] = 3
m[1, 0] = 4
m[1, 1] = 5
m[1, 2] = 6
```

Указатели на функции(1)

1. Функция является блоком кода.
2. Код находится в памяти, т. е. имеет адрес.
3. Указатель хранит адрес.

Следствие: возможен указатель на функцию.

1. Вызов функции — подготовка аргументов и передача управления по ее адресу.
2. Указатель на функцию хранит ее адрес.

Следствие: имея указатель на функцию, ее можно вызвать, даже не зная имени (но зная сигнатуру).

Вывод: библиотека может вызвать функцию программы, если программа передаст библиотеке указатель на нее.

Указатели на функции (2)

```
double sum(double x, double y) { return x + y; }
double multiply(double x, double y) { return x * y; }

// typedef double (*Op)(double x, double y);
using Op = double(double x, double y);

Op op = sum;
op(3, 4); // 7
op = multiply;
op(3, 4); // 12
```

Тип указателя на функцию с определенной сигнатурой.

Тип возвращаемого значения.

Типы параметров. Имена параметров значения не имеют.

Callback-функции

```
// matrix.h
using Op = void(Matrix* m, uint32_t i, uint32_t j);
void matrix_op(const Matrix* m, Op op);

// matrix.cpp
void matrix_op(const Matrix* m, Op op) {
        for (uint32_t i = 0; i < m->rows; i++)
                for (uint32_t j = 0; j < m->cols; j++)
                        op(m, i, j);
}

// program.cpp
cout << “Вывод элементов матрицы: " <<
        matrix_op(m, print); // print() та же, что раньше
```