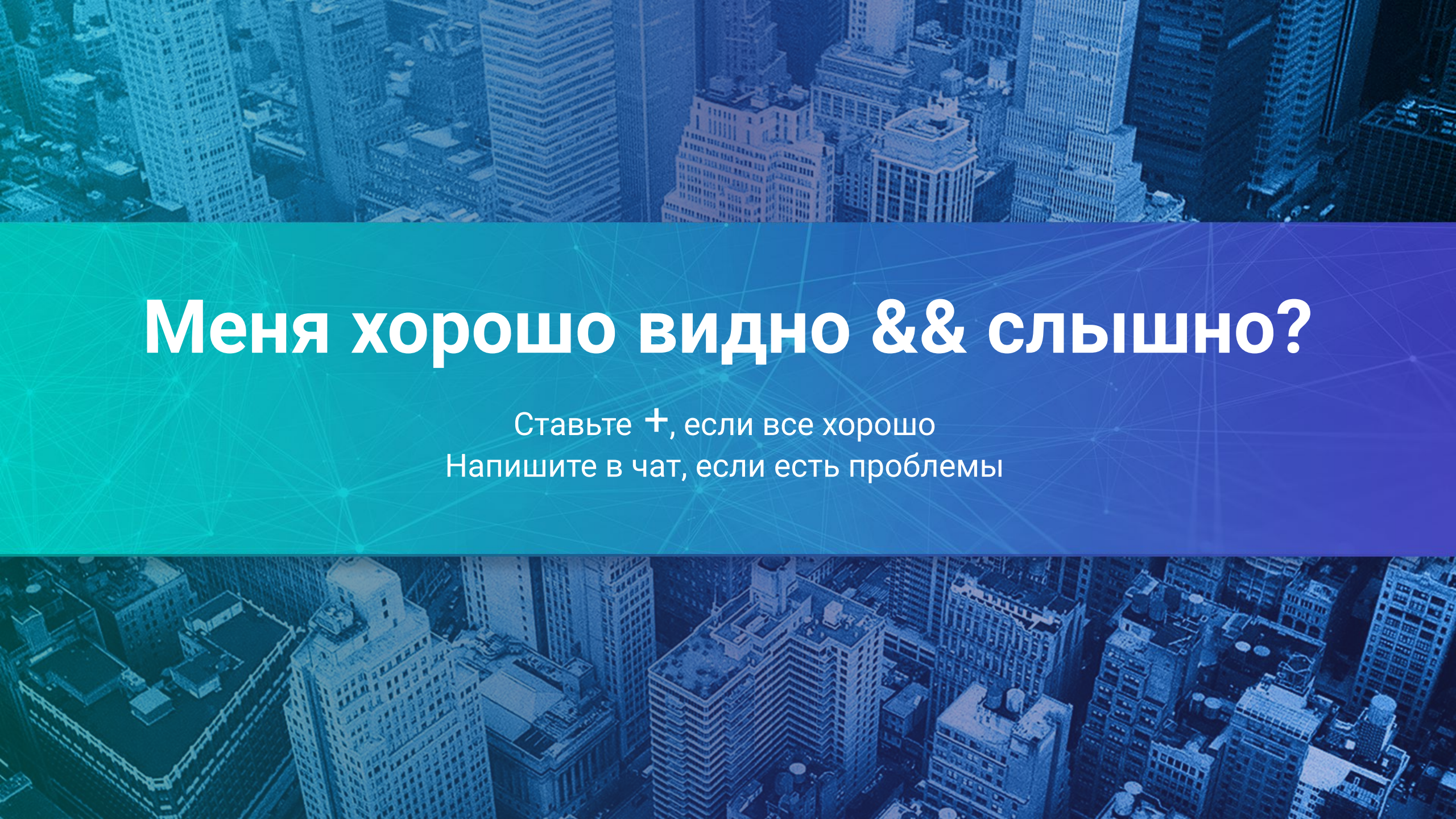

OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

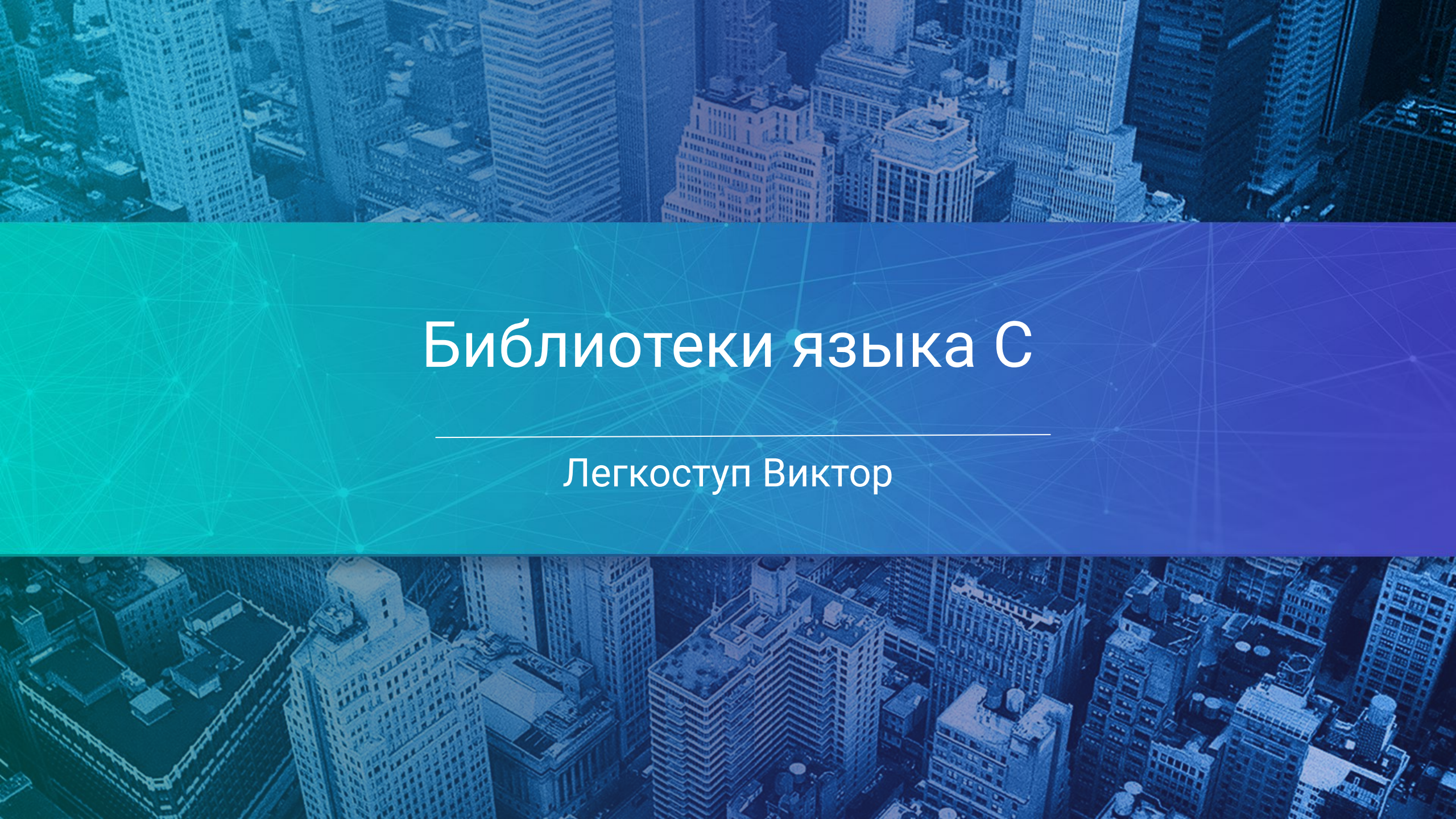


Меня хорошо видно && слышно?

Ставьте +, если все хорошо

Напишите в чат, если есть проблемы

Проверить, идет ли запись!



Библиотеки языка C

Легкоступ Виктор

Преподаватель

Легкоступ Виктор

- 7 лет работы научным сотрудником в области разработки алгоритмов управления беспилотными летательными аппаратами
- Специализация: фильтрация данных, оценивание параметров систем, системы автоматического управления, обработка сигналов, программирование микроконтроллеров, численные методы.
- Рабочие инструменты: C++, Matlab/Simulink, Mathematica, Python

Правила вебинара

Активно участвуем

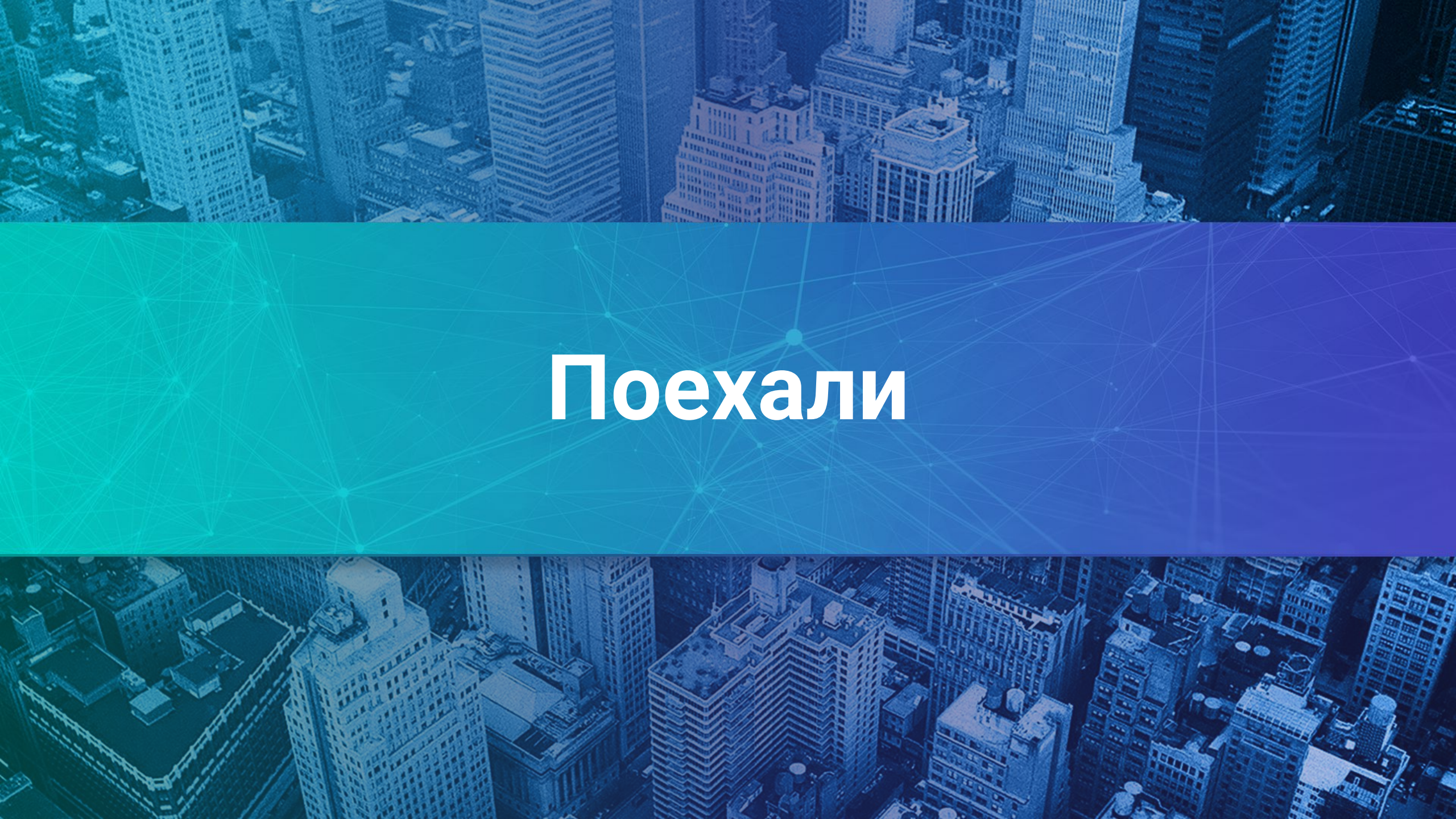
Задаем вопросы в чат или голосом

Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу

Цели и смысл вебинара | На занятии вы сможете

1. Определить, что такое библиотеки и для чего они нужны
2. Рассмотреть принципы создания библиотек
3. Создать библиотеку


Поехали

Проблемы и предпосылки

Можно хранить весь код в одном или нескольких исходных файлах проекта

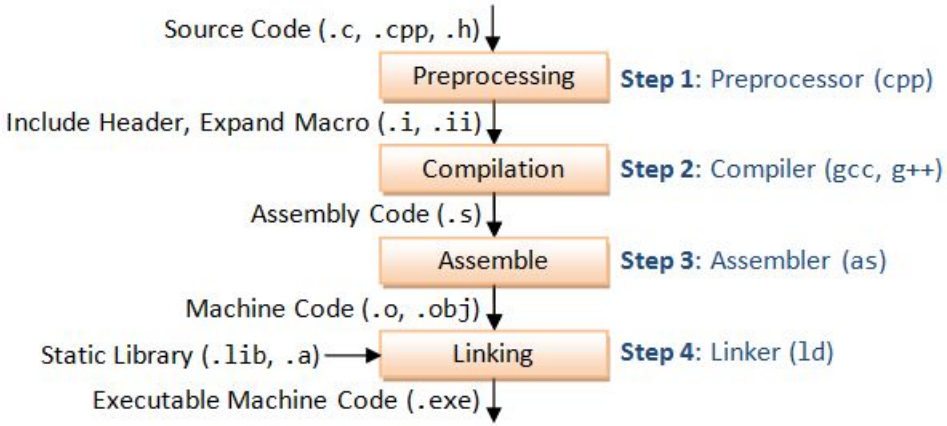
Но остается три проблемы:
а) большое время компиляции
б) низкая целостность и надежность отлаженного кода
в) сложность обновлений готового приложения

а также проблема использования одного и того же кода в разных приложениях (к примеру чтение файлов)

Какое может быть решение?

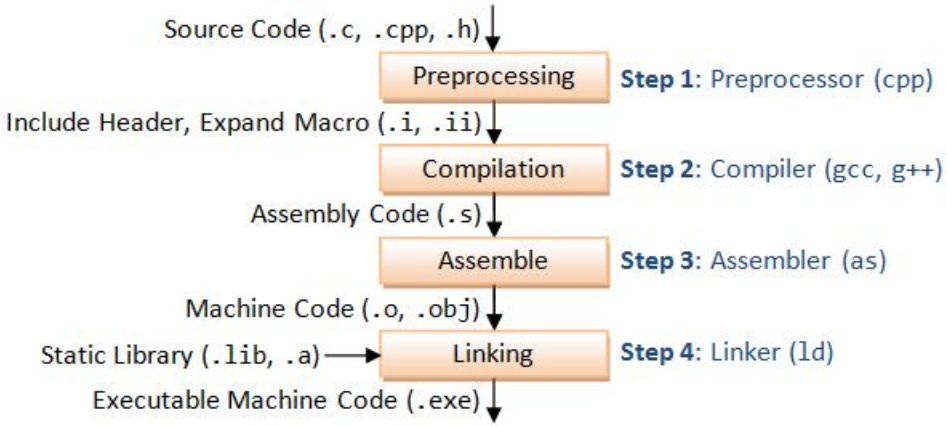
Процесс сборки программы из C-кода

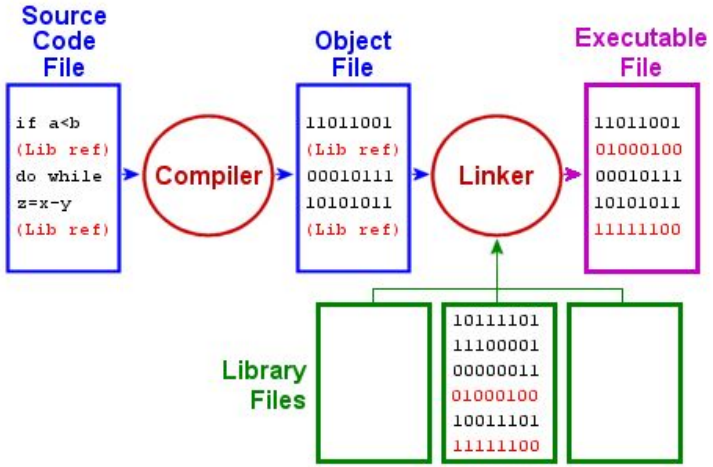
Использование только статических библиотек



Процесс сборки программы из C-кода

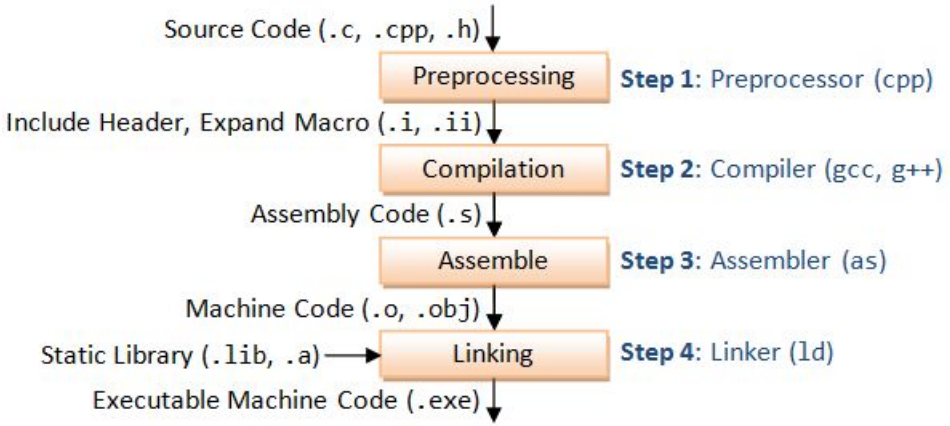
Использование только статических библиотек

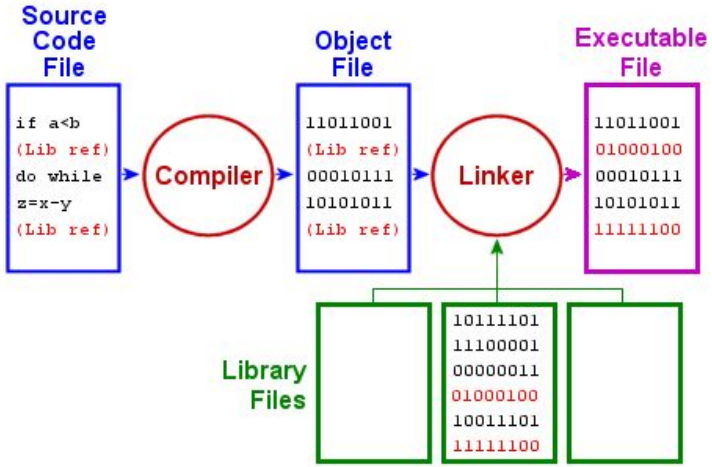




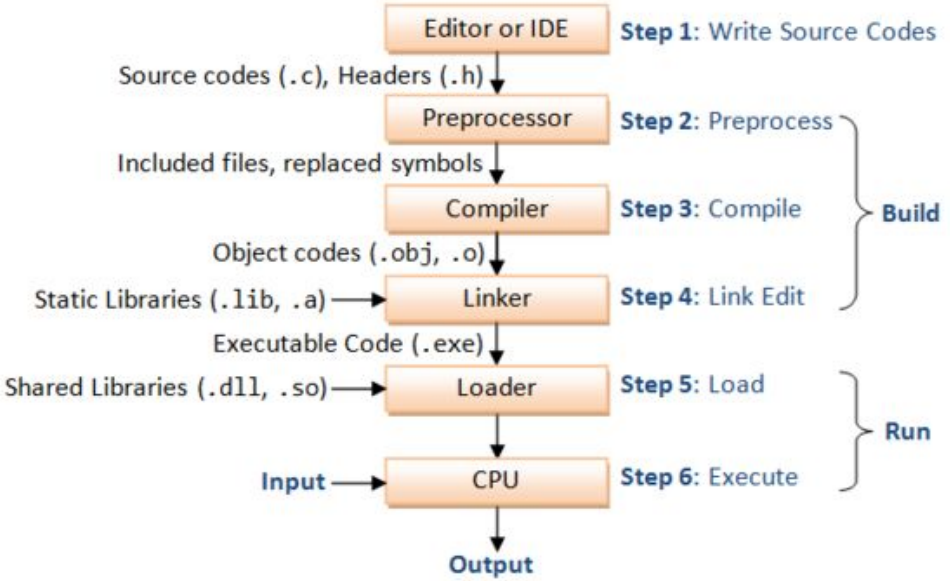
Процесс сборки программы из C-кода

Использование только статических библиотек



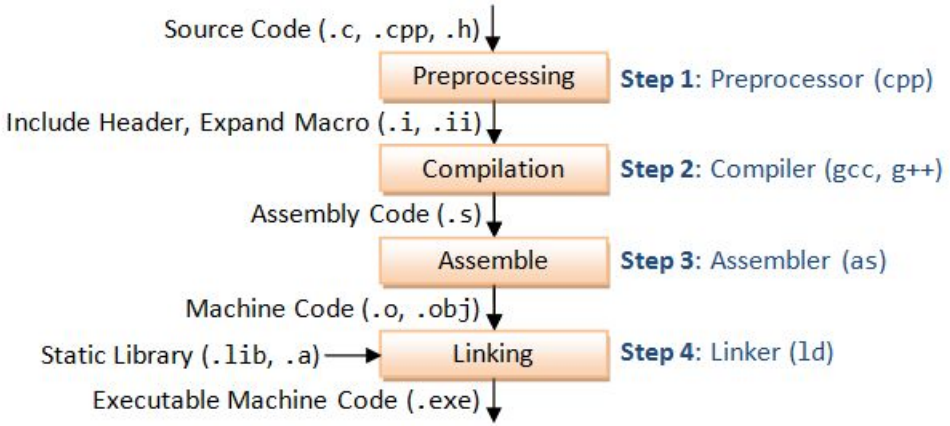


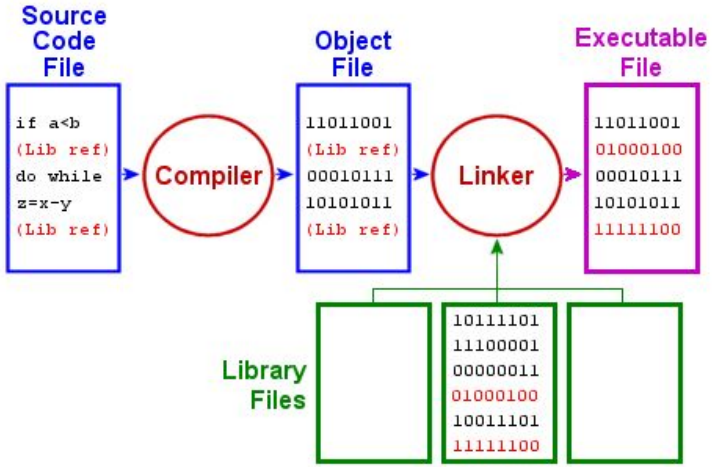
Использование статических и динамических библиотек



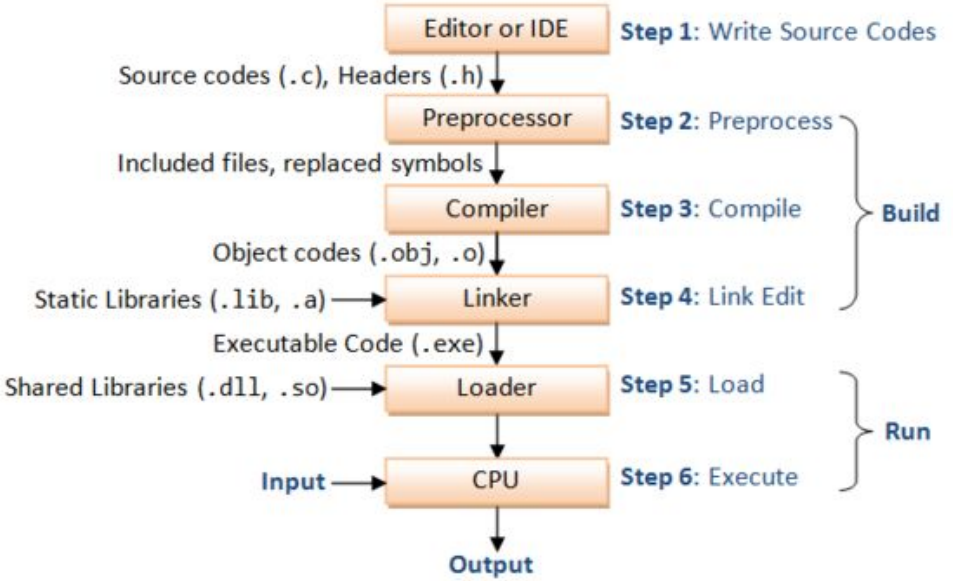
Процесс сборки программы из C-кода

Использование только статических библиотек



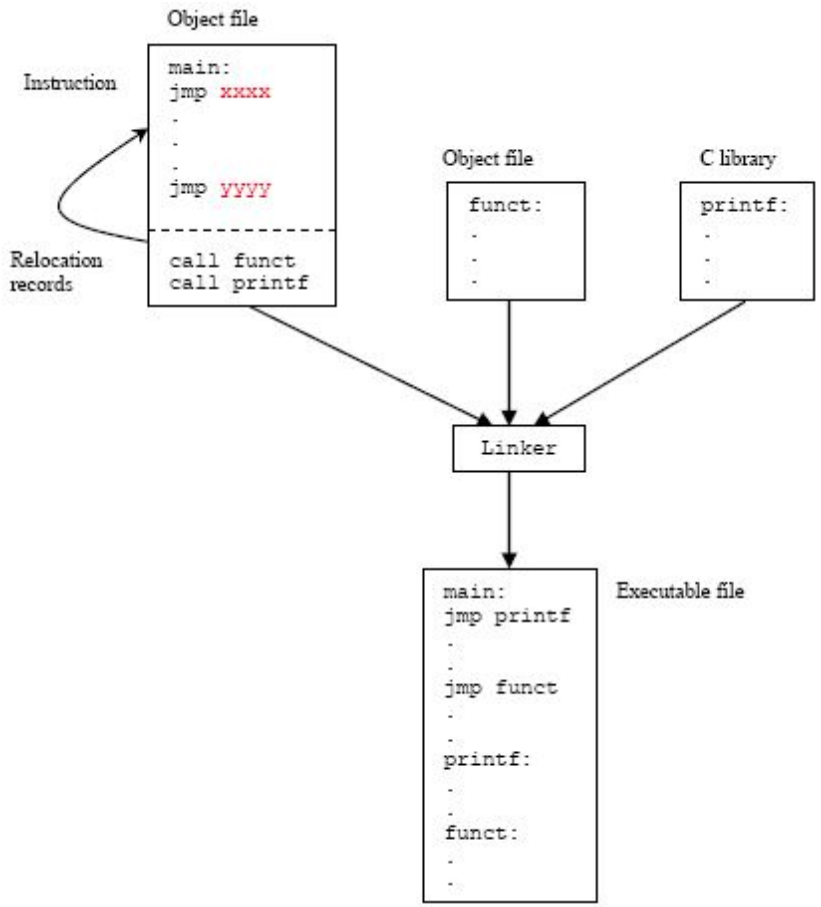


Использование статических и динамических библиотек

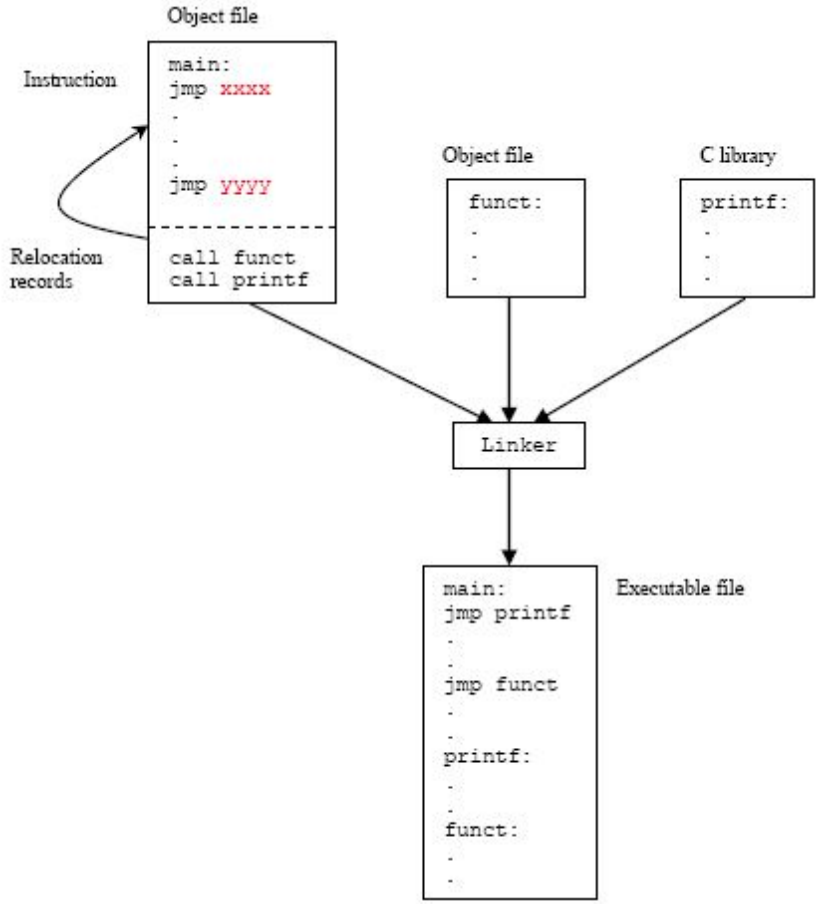


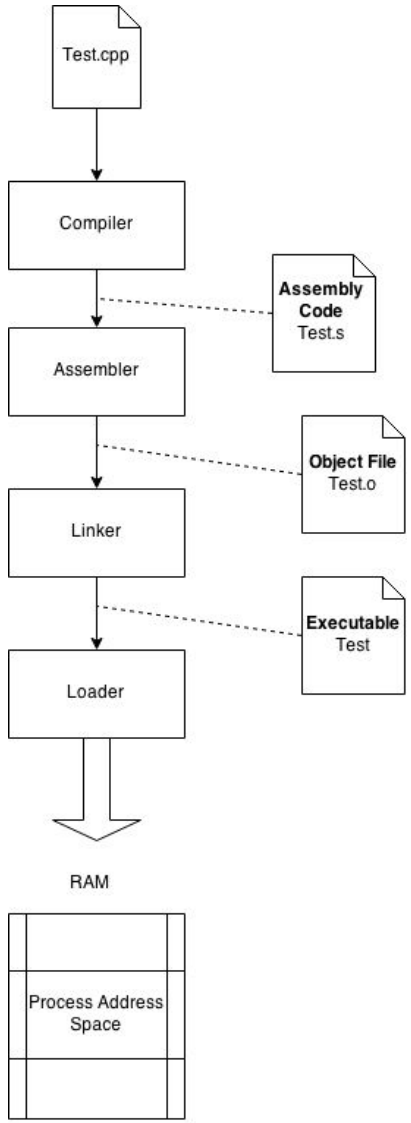
Статические библиотеки	**Динамические библиотеки**
.lib (Windows) .a (Linux, MacOS)	.dll (Windows: PE, PE32+, COM, MZ форматы) .dll.a (Windows, использование библиотеки импорта) .so (Linux: ELF, COFF форматы) .dylib (MacOS: Mach-O формат)

Процесс компоновки

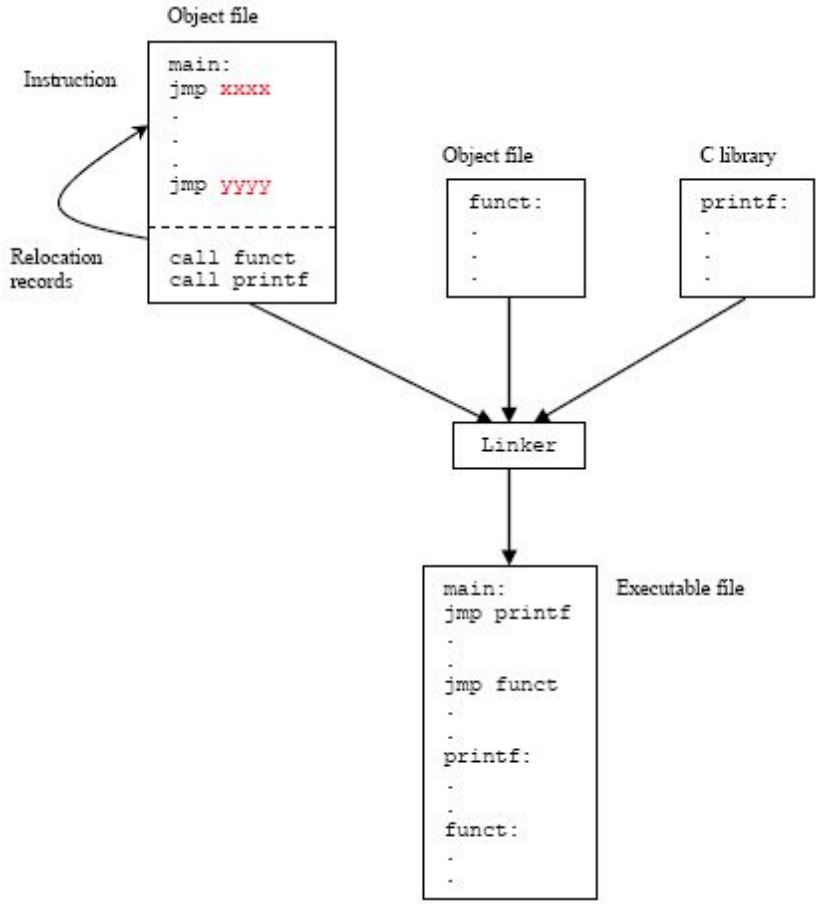


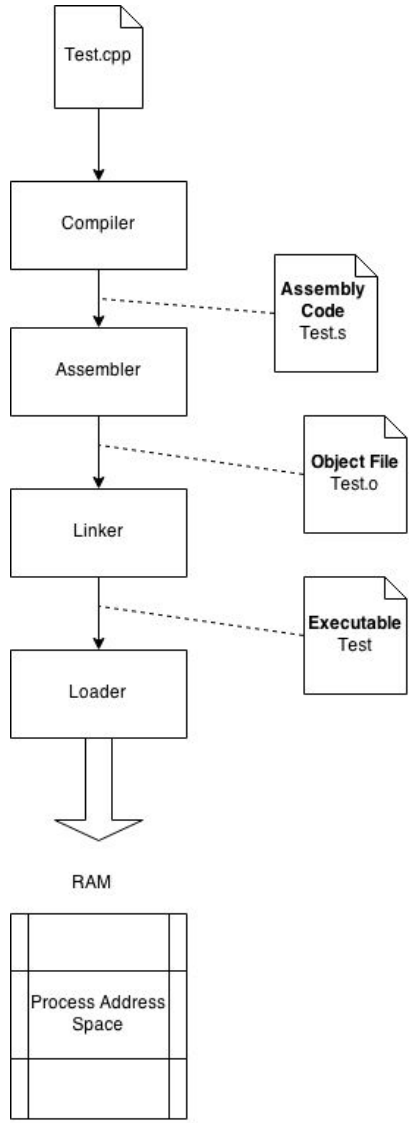
Процесс компоновки

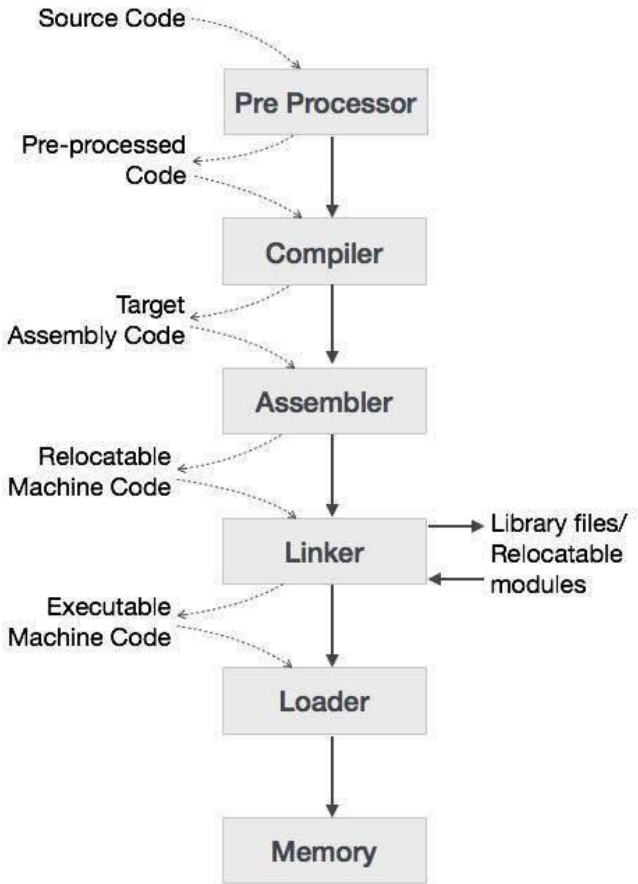




Процесс компоновки







Соглашения вызовов и искажение имен функций

Препроцессор видоизменяет имена функций, добавляя к ним идентификаторы соглашений вызовов и количество передаваемых функциям байт как аргументов. Таким образом не надо везде использовать файлы объявлений вызываемых функций – все что надо уже закодировано в имени функции. Так например обстоит дело с соглашением stdcall

Если мы хотим работать с внешними функциями, особенно динамически подключаемых библиотек, то надо как-то определиться с именами функций. К примеру мы можем указать квалификатор extern (__declspec(dllexport) для Windows) перед именем функции для предотвращения искажения её имени.

Импортируемые и экспортируемые функции (ресурсы) помещаются в соответствующие разделы объектных файлов .obj

	Cleans Stack	Arguments	Arg Ordering
cdecl	Caller	On the Stack	Right-to-left
fastcall	Callee	ECX,EDX, then stack	Left-to-Right
stdcall	Callee	On the Stack	Left-to-Right
VC++ thiscall	Callee	EDX (this), then stack	Right-to-left
GCC thiscall	Caller	On the Stack (this pointer first)	Right-to-left

Соглашения вызовов и искажение имен функций

Следующая структура кода используется для предотвращения искажений имен функций компилятором C++, реализующим возможность их перегрузки

```
#ifdef __cplusplus
extern "C"
{
#endif

// экспортируемые функции

#ifdef __cplusplus
} // __cplusplus defined.
#endif
```

Полезные утилиты

Утилиты :
ar (ranlib) – сборка стат. библиотеки,
as – ассемблер,
ld –линкер,
objdump, readelf, size, nm, strings – анализ исполняемых файлов,
gdb – отладчик.

LD_LIBRARY_PATH – тут будет производиться поиск динамических библиотек (Линукс)

Создание библиотек в GCC

Создание статической библиотеки

Скомпилировать и получить объектные файлы из исходников: gcc -c ./source/*.c

ключ (c) – создать объектный файл

Создать библиотеку - архивируем объектные файлы утилитой ar для получения статической библиотеки: ar cr libmy1.a *.o

ключ (r) – режим добавления новых файлов к архиву или создания нового архива если его нет

ключ (t) – посмотреть содержимое библиотеки (ar –t libmy1.a)

Слинковать объектные файлы программы и библиотеки: gcc *.o –Lstatic –lmy_static –o app_with_static

ключ (o) – создать исполняемый файл

ключ (L) – путь к файлам библиотеки, который следует сразу за ключом без пробела и без первого/последнего слэшей

(L. – если библиотека в этом же каталоге)

ключ (l) – линкуемые библиотеки, которые следуют сразу за ключом без пробела. Префикс lib и суффикс .a опускаются

Выполнить: ./ app_with_static

Создание динамической библиотеки

Получить объектные файлы из исходников: gcc -c -fPIC source/*.c

ключ (fPIC) – создание кода с относительной адресацией (position independent code)

Создать библиотеку: gcc -shared -o libmy2.so *.o

Слинковать: gcc *.o –Lshared –lmy_shared –o app_with_shared

Добавить путь к библиотеке: export LD_LIBRARY_PATH=$ LD_LIBRARY_PATH :”path/”

Проверить что путь появился: echo $LD_LIBRARY_PATH

Или добавить библиотеку прямо в системную директорию: sudo mv bin/shared/libtq84.so /usr/lib sudo chmod 755 /usr/lib/libtq84.so

Выполнить: ./app_with_shared

Создание библиотек в macOS

Создание статической библиотеки

Скомпилировать и получить объектные файлы из исходников: gcc -c ./source/*.c

ключ (c) – создать объектный файл

Создать библиотеку - архивируем объектные файлы утилитой ar для получения статической библиотеки: ar cr libmy1.a *.o

ключ (r) – режим добавления новых файлов к архиву или создания нового архива если его нет

ключ (t) – посмотреть содержимое библиотеки (ar –t libmy1.a)

Слинковать объектные файлы программы и библиотеки: gcc *.o –Lstatic –lmy_static –o app_with_static

ключ (o) – создать исполняемый файл

ключ (L) – путь к файлам библиотеки, который следует сразу за ключом без пробела и без первого/последнего слэшей

(L. – если библиотека в этом же каталоге)

ключ (l) – линкуемые библиотеки, которые следуют сразу за ключом без пробела. Префикс lib и суффикс .a опускаются

Выполнить: ./ app_with_static

Создание динамической библиотеки

Создание библиотеки с именем libmylib.dylib: gcc --verbose -dynamiclib -o libmy_lib.dylib my_lib.c

Подключение библиотеки, находящейся в текущей папке: gcc -L./ -lmy_lib -o app main.c

Динамическое подключение библиотеки в GCC

```
#include <dlfcn.h> // Подключаем библиотеку функций для подгрузки дин. библиотек
#include <stdio.h>
#include <stdlib.h>
// заголовочник дин. подгружаемой библиотеки добавлять НЕ надо

// где-то в main (), когда нам понадобятся функции из дин. Библиотеки
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-but-set-variable"
int (*fptr_fault) (int); // для демонстрации неверного обращения
#pragma GCC diagnostic pop
void (*fptr_my) (void); // наша функция, которую будем подтягивать в процессе выполнения программы

int main()
{
   void* dyn_lib = dlopen ("libmy_lib.dylib", RTLD_LAZY | RTLD_GLOBAL); // подгрузка библиотеки
   if (!dyn_lib) { // проверка на успешность
      printf ("Could not open libmy_shared.so \n");
      exit (1);
   }
   fptr_my = dlsym (dyn_lib, "func_from_static_lib"); // Назначаем указатель на функцию дин. библиотеки
   fptr_fault = dlsym (dyn_lib, "doesNotExist"); // Назначаем указатель на несуществующую функцию
   if (! fptr_my) { printf ("Could not get function pointer for “init_func”. Error is: %s\n\n", dlerror()); }
   if (! fptr_fault) { printf ("Could not get function pointer for non-existent function. Error is: %s\n\n", dlerror()); }
   fptr_my (); // вызываем ту самую очень нужную нам функцию
   dlclose(dyn_lib); // выгружаем библиотеку
}
```

Компилируем такую программу с библиотекой dl: gcc src/dynamic-library-loader.c -ldl -o app_with_dyn

Динамическое подключение библиотеки в MinGW

Файл «main.c»

```
#include <windows.h>        // Подключаем библиотеку Windows
// заголовочник дин. подгружаемой библиотеки добавлять НЕ надо
…
// где-то в main (), когда нам понадобятся функции из дин. библиотеки
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-but-set-variable"
        int (*fptr_fault) (int); // для демонстрации неверного обращения
#pragma GCC diagnostic pop
        void  (*fptr_my) (void); // наша функция, которую будем подтягивать в процессе выполнения программы

void* dyn_lib = LoadLibrary ("libmy_shared.dll”); // подгрузка библиотеки
if (!dyn_lib) {    // проверка на успешность
        printf ("Could not open libmy_shared.dll \n");
        exit (1);
}
fptr_my    = GetProcAddress (dyn_lib, “my_func"); // Назначаем указатель на функцию дин. библиотеки
fptr_fault = GetProcAddress (dyn_lib, "doesNotExist"); // Назначаем указатель на несуществующую функцию
if (! fptr_my) { printf ("Could not get function pointer for “init_func”. ); }
if (! fptr_fault) { printf ("Could not get function pointer for non-existent function. ); }

fptr_my (); // вызываем ту самую очень нужную нам функцию
FreeLibrary(dyn_lib);
```

Динамическое подключение библиотеки в MinGW

Файл «libmy_shared.h»

```
// остальной код

#if defined(BUILD_DLL)
#  define DLLOPT __declspec(dllexport)
#else
#  define DLLOPT __declspec(dllimport)
#endif

#ifdef __cplusplus
extern "C" {
#endif

    void DLLOPT my_func(void);

#ifdef __cplusplus
}
#endif

// остальной код
```

При компиляции библиотеки добавить ключи -BUILD_DLL=1 и –DWIN32 и -Wl,--subsystem,windows
Для получения dll также добавить --out-implib,libadd.a

Динамическое подключение библиотеки в MinGW

Файл «libmy_shared.c»

```
#include "libmy_shared.h"

BOOL APIENTRY DllMain(HINSTANCE hinstDLL,
    DWORD fdwReason, LPVOID lpvReserved)
{
  switch (fdwReason)  {
   case DLL_PROCESS_ATTACH:  // Подключение DLL
    // код
    break;

   case: DLL_PROCESS_DETACH: // Отключение DLL
    // код
    break;

   case DLL_THREAD_ATTACH: // Создание нового потока
    // код
    break;

   case DLL_THREAD_DETACH: // Завершение потока
    // код
    break;
  }
}

void DLLOPT my_func(void) {
        // что-то делаем
}
```

https://malicious.link/post/2020/compiling-a-dll-using-mingw/
https://www.transmissionzero.co.uk/computing/building-dlls-with-mingw/

Динамическое подключение библиотеки в MinGW

Вызвав утилиту DumpBin.exe -exports из пакета Visual Studio можно посмотреть раздел экспорта библиотеки
Вызвав утилиту DumpBin.exe -imports из пакета Visual Studio можно посмотреть раздел импорта программы

Soname
Позволяет задавать версии используемых библиотек, тем самым вручную указывая совместимость/несовместимость версий

Источники

Создание библиотек под Windows:
https://www.transmissionzero.co.uk/computing/building-dlls-with-mingw/
https://malicious.link/post/2020/compiling-a-dll-using-mingw/
Джеффри Рихтер – Windows для профессионалов.

Создание библиотек под Linux:
Linux. Системное программирование.
UNIX. Профессиональное программирование.
http://www.opennet.ru/docs/RUS/zlp/