АКАДЕМИЯ АЙТИ

Работа с библиотеками

Дубров Денис Владимирович
Ведущий инженер-программист, к.ф.-м.н.
denis.dubrov@harman.com

Модульный подход к проектированию ПО

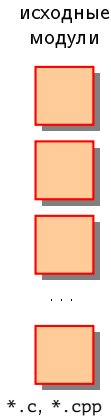


Рис.: двухэтапное построение исполняемого модуля

Модульный подход к проектированию ПО

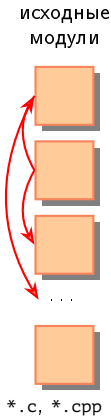


Рис.: двухэтапное построение исполняемого модуля

Модульный подход к проектированию ПО

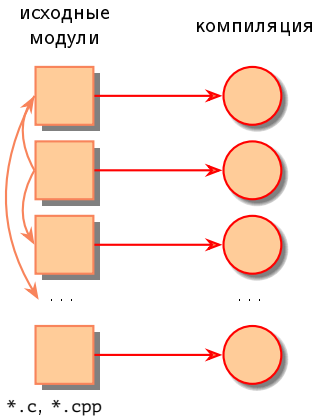


Рис.: двухэтапное построение исполняемого модуля

Модульный подход к проектированию ПО

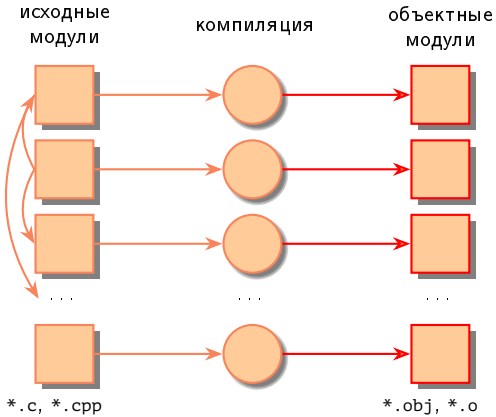


Рис.: двухэтапное построение исполняемого модуля

Модульный подход к проектированию ПО

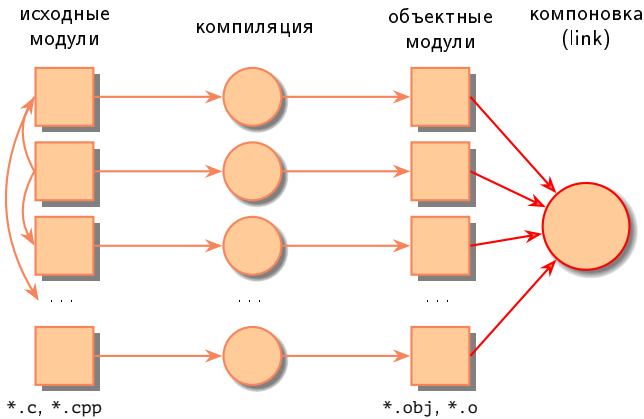


Рис.: двухэтапное построение исполняемого модуля

Модульный подход к проектированию ПО

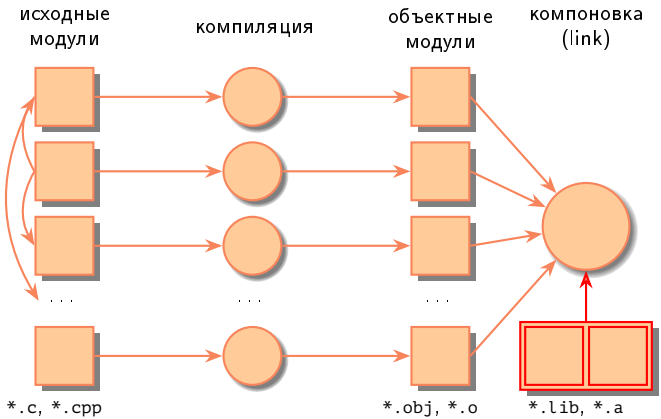


Рис.: двухэтапное построение исполняемого модуля

Модульный подход к проектированию ПО

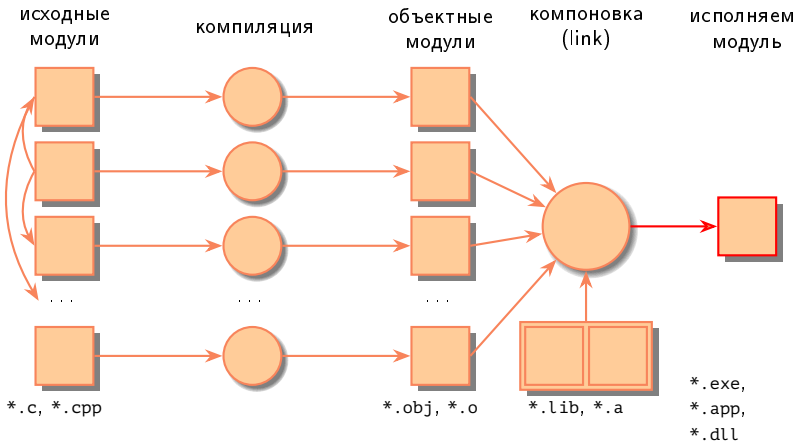


Рис.: двухэтапное построение исполняемого модуля

Объявления и определения

Определение (объявление)

Объявление (declaration) — вводит новые имена в транслируемый модуль или повторно вводит имена, введённые в предыдущих объявлениях. Может быть одновременно **определением (definition)** этих имён.

Пример (объявление)

```
void inc(int &rn);
```

Пример (определение)

```
void inc(int &rn)
{
  ++ rn;
}
```

Область действия и связь

Определение (область действия и связь)

Область действия (scope) — для имени — часть программы, в которой оно может использоваться и означает одно и то же.

Связь (linkage) — свойство имени, позволяющее означать то же самое в другой области, если в ней дать его (повторное) объявление.

Применение средств модульного программирования

f.cpp

```
// ...

extern void f()
{
  // ...
}
```

g.cpp

```
// ...

void g()
{
  // ...
  f();
  // ...
}
```

Применение средств модульного программирования

f.cpp

```
// ...

void f()
{
  // ...
}
```

g.cpp

```
// ...

void g()
{
  // ...
  f();
  // ...
}
```

Применение средств модульного программирования

f.cpp

```
// ...

void f()
{
  // ...
}
```

g.cpp

```
void f();

// ...

void g()
{
  // ...
  f();
  // ...
}
```

Применение средств модульного программирования

f.cpp

```
// ...

void f()
{
  // ...
}
```

f.h

```
void f();
```

g.cpp

```
#include "f.h"

// ...

void g()
{
  // ...
  f();
  // ...
}
```

Правила создания заголовочных файлов

Правила

- Помещаются объявления и определения, необходимые в нескольких транслируемых модулях.
- Стандартное расширение (h, hpp).
- В начале и конце вставляются «охранные» директивы препроцессора, предотвращающие включение содержимого файла при его повторном подключении:

Правила создания заголовочных файлов (пример)

Пример

```
/*
  mydefs.h: My definitions
*/

#ifndef MYDEFS_H__
#define MYDEFS_H__

/* ... */

#endif    /* MYDEFS_H__ */
```

Правила создания заголовочных файлов (продолжение)

Правила (продолжение)

» Объявления и определения должны быть идентичными для всех транслируемых модулей.

Пример (b.h)

```
struct Data
{
  int m_n1;
#ifdef INCLUDED_FROM_B_CPP
  int m_n2, m_n3, m_n4;
#endif
};
```

Пример (b.h, окончание)

```
void f(Data d);
```

Правила создания заголовочных файлов (продолжение)

a.cpp

```cpp
#include "b.h"

int main()
{
  Data d;
  f(d);
  // ...
}
```

b.cpp

```cpp
#define INCLUDED_FROM_B_CPP
#include "b.h"

void f(Data d)
{
  // ...
  d.m_n4 = 4;
}
```

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

» Макроопределения.

Пример

```
#define PI 3.1415926
```

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

- Макроопределения.
- Объявления типов, для которых не нужно полное определение.

Пример

```
struct Hidden;

struct Data
{
  // ...
  Hidden *m_pHidden;
};
```

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

- Макроопределения.
- Объявления типов, для которых не нужно полное определение.
- Определения всех остальных типов.

Пример

```
struct Hidden;

struct Data
{
  // ...
  Hidden *m_pHidden;
};
```

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

- Макроопределения.
- Объявления типов, для которых не нужно полное определение.
- Определения всех остальных типов.
- Объявления функций.

Пример

```
double my_fast_sin(double d);
```

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

- Макроопределения.
- Объявления типов, для которых не нужно полное определение.
- Определения всех остальных типов.
- Объявления функций.
- Директивы подключения заголовочных файлов с описаниями, необходимыми для описаний текущего файла.

Пример

```c
#include <stddef.h> // size_t

size_t get_size();
```

Объекты, помещаемые, в заголовочные файлы (C++)

Помещаемые объекты

» Определения констант.

Пример

```
const double cdPi = 3.1415926;
```

Объекты, помещаемые, в заголовочные файлы (C++)

Помещаемые объекты

- Определения констант.
- Определения встроенных функций (**inline**).

Пример

```
inline void Inc(int &rn)
{
  ++ rn;
}
```

Объекты, помещаемые, в заголовочные файлы (C++)

Помещаемые объекты

- Определения констант.
- Определения встроенных функций (**inline**).
- Объявления шаблонов, для которых не нужно полное определение.

Пример

```
template <class T>
  struct Hidden;

template <class T>
  struct Data
{
  // ...
  Hidden <T> *m_pHidden;
};
```

Объекты, помещаемые, в заголовочные файлы (C++)

Помещаемые объекты

- Определения констант.
- Определения встроенных функций (**inline**).
- Объявления шаблонов, для которых не нужно полное определение.
- Определения всех остальных шаблонов.

Пример

```
template <class T>
  struct Hidden;

template <class T>
  struct Data
{
  // ...
  Hidden <T> *m_pHidden;
};
```

Объекты, помещаемые, в заголовочные файлы (C++)

Помещаемые объекты

- Определения констант.
- Определения встроенных функций (**inline**).
- Объявления шаблонов, для которых не нужно полное определение.
- Определения всех остальных шаблонов.
- Пространства имён (вложены) кроме макроопределений.

Пример

```
namespace mylib
{
  const double cdPi = 3.1415926;
  //
  double my_fast_sin(double d);
  //
  // ...
}
```

Правила для пространств имён

Замечания

- Пространство имён `std` зарезервировано для стандартной библиотеки, в нём нельзя ничего объявлять.
- Не принято помещать в заголовочные файлы директивы **`using`** (**`using namespace`** `x;`), также в ограниченных случаях следует использовать объявления **`using`** (**`using`** `x::f;`).

Языковая связь

extern ⟨*строковый_литерал*⟩ ⟨*объявление*⟩
extern ⟨*строковый_литерал*⟩ { ⟨*объявления*⟩ }

Пример

```
extern "C++" int f(int);

extern "C"
{
  int fInt(int);
  double fDouble(double);
}
```

Типы

C/C++	Fortran
bool (C++)	**logical***1
unsigned char	byte
short int	**integer***2
int или **long**	**integer**, **logical**
float	**real**
double	**real***8
long double	**real***16

C/C++	Fortran
std::complex <**float**>	**complex**
std::complex <**double**>	**double complex**
type [M][N]	type name(M, N)
char [N]	**character***N name
char [N][M]	**character***N name(M)

Таблица: соответствие типов C++ / Fortran

Расположение массивов в памяти

C /C++:

```
a(1,1)  a(2,1)  ...  a(M,1)    ...  a(1,N)    a(2,N)    ...  a(M,N)
```

Fortran:

```
a[0,0]  a[0,1]  ...  a[0,N-1]  ...  a[M-1,0]  a[M-1,1]  ...  a[M-1,N-1]
```

Рис.: размещение массивов в C++ / Fortran

Вызовы функций из C++

Example (cpp_main.cpp)

```
#include <iostream>

extern "C" void f_fortran_(int *pn, float *pf, float *pfs);

int main()
{
  int n = 5;
  float af[5] = { 1., 1., 1., 1., 1. };
  float s;
  f_fortran_(&n, af, &s);
  std::cout << s << std::endl;
}
```

Определения функций в Fortran

Example (`fortran_func.f90`)

```
subroutine f_fortran(n, a, s)
  integer n
  real a(*)
  real s
  integer i
  s = 0
  do i = 1, n
    s = s + a(i)
  enddo
end subroutine f_fortran
```

Вызовы функций из C++

Example (`cpp_main.cpp`)

```
$ gfortran -c fortran_func.f90
$ g++ -c cpp_main.cpp
$ g++ cpp_main.o fortran_func.o -o prog
$ ./prog
5
```

Определения функций в Fortran

Example (`fortran_main.f90`)

```
program call_cpp
  integer i
  common/global/ i
  integer k
  i = 3
  k = 2
  call cpp_func(k)
  print*, i
end
```

Вызовы функций из C++

Example (`cpp_func.cpp`)

```
extern "C" struct
{
  int i;
}
  global_;

extern "C" void cpp_func_(int *pn)
{
  global_.i *= *pn;
}
```

Вызовы функций из C++

Example (`cpp_main.cpp`)

```
$ g++ -c cpp_func.cpp
$ gfortran -c fortran_main.f90
$ gfortran fortran_main.o cpp_func.o -o prog2
$ ./prog2
           6
```

Виды библиотек

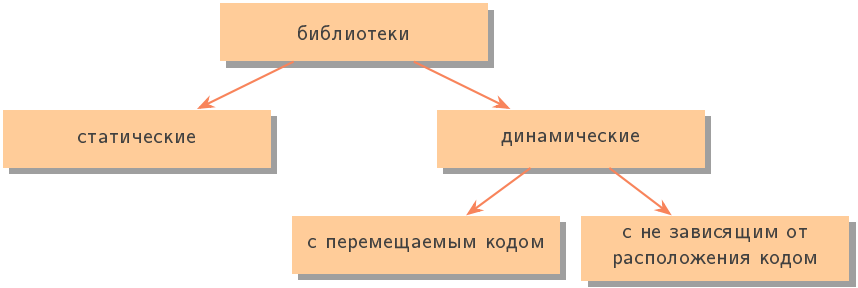


Рис.: виды библиотек

Подключение разделяемых библиотек

- статическое подключение при компоновке;
- динамическая загрузка во время выполнения.

Создание статической библиотеки

Пример

```
$ g++ -c sample1.cpp
$ g++ -c sample2.cpp
$ ar -crs libsample.a sample1.o sample2.o
$ g++ -c use-static.cpp
$ g++ -o prog1 -L. use-static.o -lsample
$ ./prog1
...
```

Создание разделяемой библиотеки в ОС Linux

Пример

```
$ g++ -fPIC -c sample1.cpp
$ g++ -fPIC -c sample2.cpp
$ g++ -shared -o libsample.so.0.0.1 sample1.o sample2.o
$ file libsample.so.0.0.1
libsample.so.0.0.1: ELF 64-bit LSB shared object, x86-64, ...
dynamically linked, ...
$ ln -s libsample.so.0.0.1 libsample.so
$ g++ -c use-static.cpp
$ g++ -L. -o prog2 use-static.o -lsample
$ LD_LIBRARY_PATH=. ./prog2
...
```

Определение зависимостей в ОС Linux

Пример

```
$ ldd llc
  linux-vdso.so.1 => ...
  libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 ...
  libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 ...
  libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 ...
  libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 ...
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 ...
  /lib64/ld-linux-x86-64.so.2 ...
```

Определение зависимостей в ОС Windows

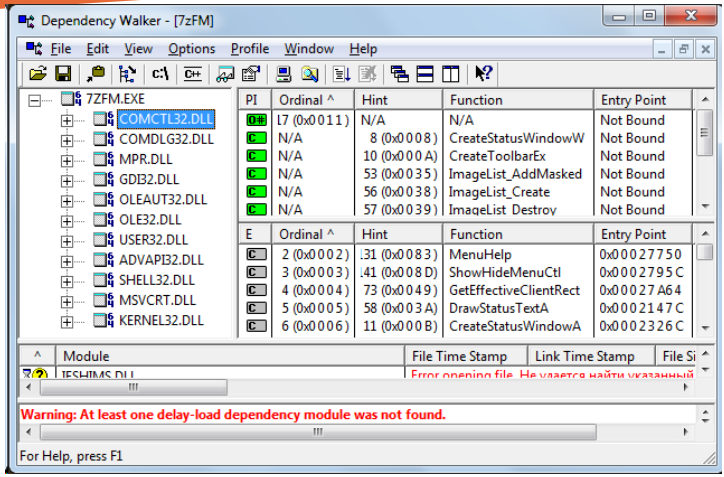


Рис.: окно программы Dependency Walker

Интерфейс плагина

Пример (interface.hpp)

```
#pragma once

struct Interface
{
  virtual int f1(int n) = 0;
};

using pfn_plugin = Interface *(*)();
```

Реализация плагина

Пример (plugin1.cpp)

```
#include "interface.hpp"

struct Plugin1 : public Interface
{
  virtual int f1(int n) override;
};

int Plugin1::f1(int n)
{
  return n + 1;
}
```

Пример (plugin1.cpp, конец)

```
extern "C" Interface *plugin_func()
{
  static Plugin1 plugin;
  return &plugin;
}
```

Использование плагина

Пример (use-plugin.cpp)

```
#include "interface.hpp"
#include <dlfcn.h>
#include <iostream>

int main()
{
  void *pvLib = dlopen("./plugin1.so", RTLD_LAZY);
  if (!pvLib)
  {
    std::cerr << dlerror() << std::endl;
    return EXIT_FAILURE;
  }
```

Использование плагина (конец)

Пример (use-plugin.cpp, конец)

```
  pfn_plugin pfnEntry;
  *(void **) (&pfnEntry) = dlsym(pvLib, "plugin_func");
  if (!pfnEntry)
  {
    std::cerr << dlerror() << std::endl;
    return EXIT_FAILURE;
  }
  Interface *pPlugin = (*pfnEntry)();
  std::cout << pPlugin->f1(1) << std::endl;
  dlclose(pvLib);
}    // main()
```

Сборка примера

Пример

```
$ g++ -fPIC -c plugin1.cpp
$ g++ -shared -o plugin1.so plugin1.o
$ g++ -o use-plugin use-plugin.cpp -ldl
$ ./use-plugin
2
```

Макросы экспорта символов

Пример (export-lib.h)

```
#if defined _WIN32 || defined __CYGWIN__
  #ifdef BUILDING_DLL
    #ifdef __GNUC__
      #define DLL_PUBLIC __attribute__ ((dllexport))
    #else
      #define DLL_PUBLIC __declspec(dllexport)
    #endif
  #else
    #ifdef __GNUC__
      #define DLL_PUBLIC __attribute__ ((dllimport))
    #else
      #define DLL_PUBLIC __declspec(dllimport)
```

Макросы экспорта символов (конец)

Пример (export-lib.h, конец)

```
    #endif
  #endif
  #define DLL_LOCAL
#else
  #if __GNUC__ >= 4
    #define DLL_PUBLIC __attribute__ ((visibility ("default")))
    #define DLL_LOCAL  __attribute__ ((visibility ("hidden")))
  #else
    #define DLL_PUBLIC
    #define DLL_LOCAL
  #endif
#endif
```

Управление видимостью символов

Пример (lib.h)

```
#pragma once

#include "export-lib.h"

DLL_PUBLIC int f1(int n);

struct DLL_PUBLIC Sample
{
  int f1(int n);
private:
  DLL_LOCAL int private_f1(int n);
};
```

Сборка примера

Пример

```
$ g++ -fPIC -fvisibility=hidden -D BUILDING_DLL -c lib.cpp
$ g++ -shared -o libsample.so lib.o
$ nm -C -D libsample.so
0000000000201020 B __bss_start
                 w __cxa_finalize
...
000000000000059a T f1(int)
00000000000005aa T Sample::f1(int)
$ g++ -o use use.cpp libsample.so
$ LD_LIBRARY_PATH=. ./use
2 4
```

Проблемы разделяемых библиотек

Проблемы при использовании разделяемых библиотек (DLL Hell)

- Перезапись общих/системных библиотек в ранних версиях Windows при установке ПО;
- Зависимость работы ПО от системных настроек (`PATH`, `LD_LIBRARY_PATH`, ...);
- Удаление библиотек из системы;
- ...

Схема работы

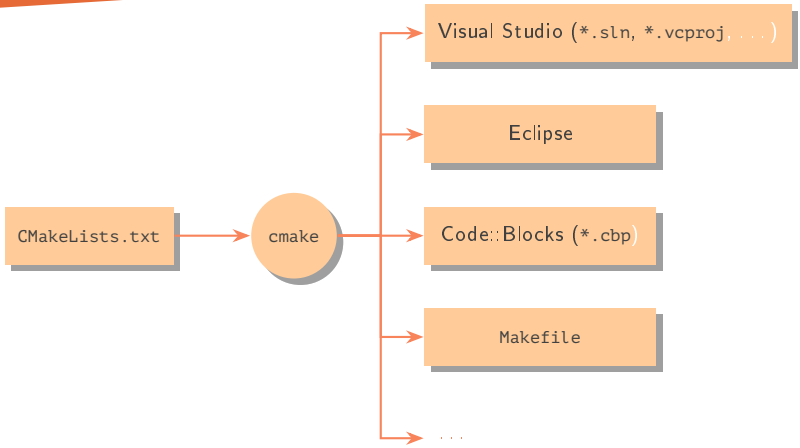


Рис.: генерация проектов при помощи утилиты CMake

Структура проекта

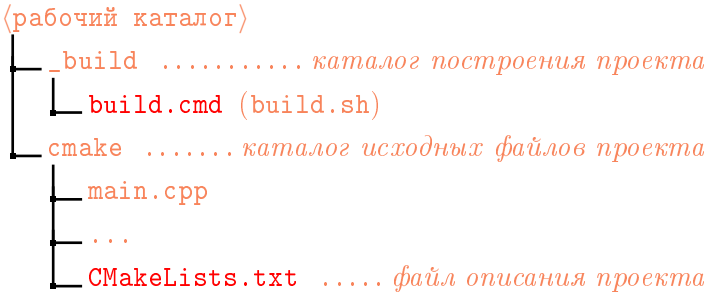


Рис.: структура каталога простого проекта

Пример

Пример (build.cmd, использование переменной окружения PATH)

```
@set PATH=C:\Program Files (x86)\CodeBlocks\MinGW\bin;%PATH%

cmake -G "CodeBlocks - MinGW Makefiles" ../test_cmake
```

Пример (build.cmd, использование переменной CMake CMAKE_PREFIX_PATH)

```
@set PATH=C:\Qt\Qt5.7.0\Tools\mingw530_32\bin;%PATH%

cmake^
  -G "MinGW Makefiles"^
  -D CMAKE_PREFIX_PATH="C:\Qt\Qt5.7.0\5.7\mingw53_32"^
  ..\qt-examples-2
```

Пример

Пример (CMakeLists.txt)

```
# CMakeLists.txt для тестового проекта

project(test_cmake)

add_executable(test_cmake main.cpp)

# Конец файла
```

Пример (продолжение)

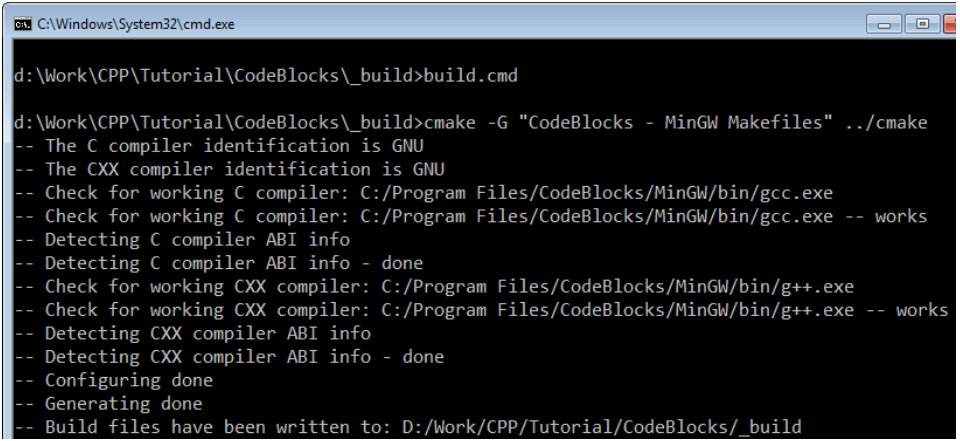
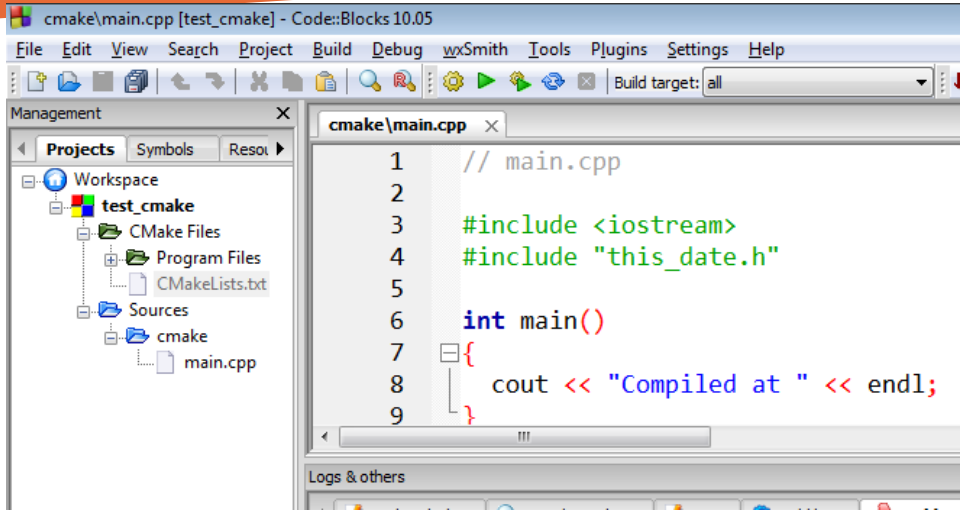
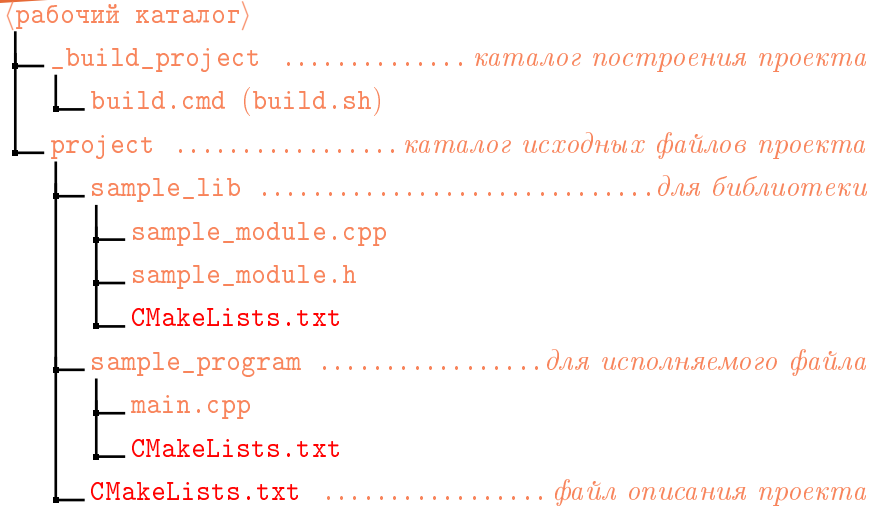


Рис.: вывод программы CMake

Пример (продолжение)



Структура проекта



Пример

Пример (CMakeLists.txt)

```
cmake_minimum_required(VERSION 2.8)

project(sample)

add_subdirectory(sample_lib)
add_subdirectory(sample_program)
```

Пример

Пример (sample_lib/CMakeLists.txt)

```
add_library(sample_lib sample_module.cpp sample_module.h)
```

Пример (sample_program/CMakeLists.txt)

```
add_executable(sample_program main.cpp)

include_directories(../sample_lib)

target_link_libraries(sample_program sample_lib)
```

Пример

Пример (`build.cmd`)

```
cmake -G "CodeBlocks - MinGW Makefiles" -DBUILD_SHARED_LIBS=0 ../project
```

Пример

Пример (build.cmd)

```
cmake -G "CodeBlocks - MinGW Makefiles" -DBUILD_SHARED_LIBS=0 ../project
```

Пример (sample_lib/CMakeLists.txt)

```
set(BUILD_SHARED_LIBS FALSE)
```

Пример

Пример (build.cmd)

```
cmake -G "CodeBlocks - MinGW Makefiles" -DBUILD_SHARED_LIBS=0 ../project
```

Пример (sample_lib/CMakeLists.txt)

```
set(BUILD_SHARED_LIBS FALSE)
```

Пример (sample_lib/CMakeLists.txt)

```
project(sample_lib)

add_library(sample_lib STATIC sample_module.cpp sample_module.h)
```

Сборка разных видов библиотек

Пример (`CMakeLists.txt`)

```
add_library(mylib1 STATIC ...)
add_library(mylib2 SHARED ...)
add_library(mylib3 MODULE ...)
```

Описание проекта со скрытыми символами

Пример (CMakeLists.txt)

```
cmake_minimum_required(VERSION 3.0)

project(mylib_and_prog)

include(GenerateExportHeader)

include_directories(${CMAKE_CURRENT_BINARY_DIR})

add_library(mylib mylib.cpp mylib.h)
generate_export_header(mylib)
add_executable(myprogram myprogram.cpp)
target_link_libraries(myprogram mylib)
```

Заголовок библиотеки со скрытыми символами

Пример (mylib.h)

```
#pragma once

#include "mylib_export.h"

int MYLIB_EXPORT f1(int n);
```

Сборка проекта со скрытыми символами

Пример

```
$ cmake -D BUILD_SHARED_LIBS=ON ~/path/to/sources
$ make
$ ./myprogram
...
```




АКАДЕМИЯ АЙТИ

Спасибо за внимание!

Центральный офис:
Москва, Варшавское шоссе 47,
корп.4, 10 этаж
Тел: +7 (495) 662-7894, 662-7895
Факс: +7(495) 974-7990
e-mail: academy@it.ru

www.academy.it.ru