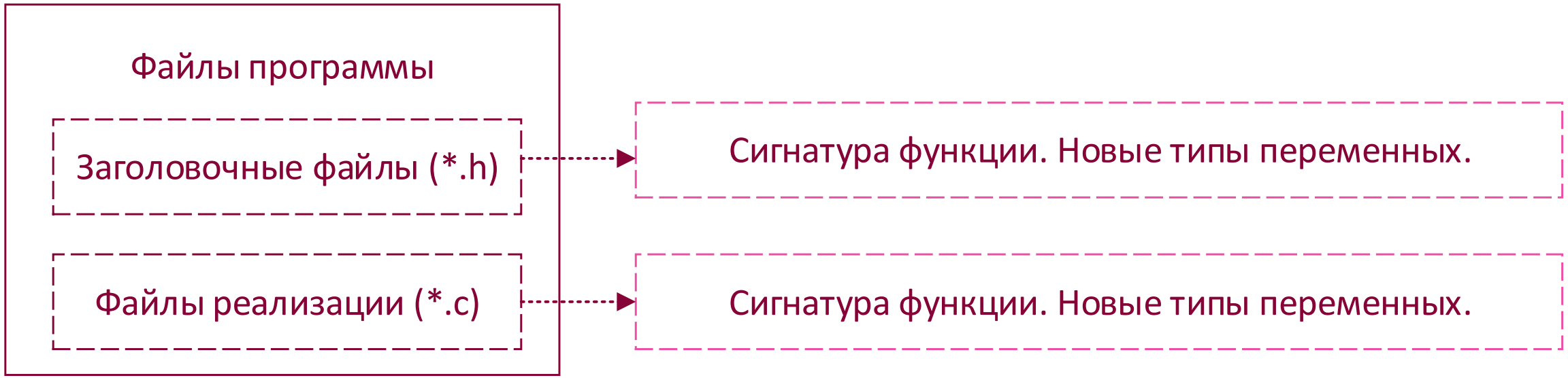
СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ЛЕКЦИЯ № 4

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.

ФАЙЛЫ ПРОГРАММЫ НА ЯЗЫКЕ С



ПРИМЕР

project

- pony.h
- pony.c
- main.c

pony.h

```
#pragma once

typedef struct {
        int age;
        char* name;
} PONY;

void add_pony(PONY** pony);
void print_pony_info(PONY* pony);
void release_pony_info(PONY* pony);
```

pony.c

```
#include <stdio.h>
#include "pony.h"

void add_pony(PONY** pony)
{
        const int name_size = 10;
        *pony = (PONY*)calloc(1, sizeof(PONY));
        (*pony)->age = rand() % 100;
        (*pony)->name = (char*)calloc(name_size, sizeof(char));
        for (int i = 0; i < name_size; i++) {
                (*pony)->name[i] = '0' + rand() % 72;
        }
        (*pony)->name[name_size - 1] = 0;
}

void print_pony_info(PONY* pony)
{
        printf("This pony's name is '%s'\tShe is %d years old.\n",
                pony->name, pony->age);
}

void release_pony_info(PONY* pony)
{
        free(pony->name);
        free(pony);
}
```

Почему в одном случае используем < >, а в другом " "?
Как в данном примере:

```
#include <stdio.h>
#include "pony.h"
```

ПРИМЕР

```c
#include "pony.h"
#define COUNT 5

int main()
{
        PONY** herd = NULL;

        srand(time(NULL));
        herd = (PONY**)calloc(COUNT, sizeof(PONY*));
        for (int i = 0; i < COUNT; i++) {
                add_pony(&herd[i]);
        }

        for (int i = 0; i < COUNT; i++) {
                print_pony_info(herd[i]);
        }

        for (int i = 0; i < COUNT; i++) {
                release_pony_info(herd[i]);
        }
        free(herd);
}
```

main.c

Используемые директивы:

- #include
- #define
- #pragma once

Аналогичны ли заголовочные файлы?

```c
#ifndef PONY_HEADER
#define PONY_HEADER

typedef struct {
        int age;
        char* name;
} PONY;

void add_pony(PONY** pony);
void print_pony_info(PONY* pony);
void release_pony_info(PONY* pony);

#endif
```

pony1.h

```c
#pragma once

typedef struct {
        int age;
        char* name;
} PONY;

void add_pony(PONY** pony);
void print_pony_info(PONY* pony);
void release_pony_info(PONY* pony);
```

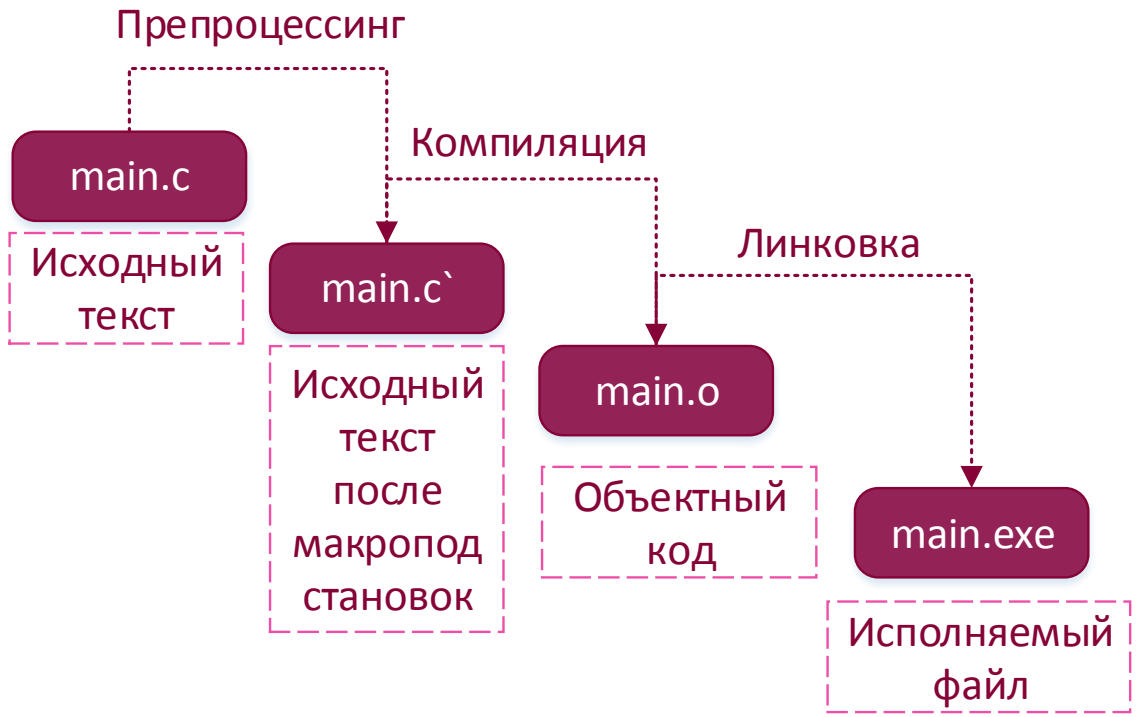
pony.h

ПРОЦЕСС КОМПИЛЯЦИИ

Процесс преобразования программы, составленной на исходном языке высокого уровня, в аналогичную программу на низкоуровневом языке, который близок к машинному коду.

Этапы компиляции

- Препроцессинг
- Компиляция
- Линковка

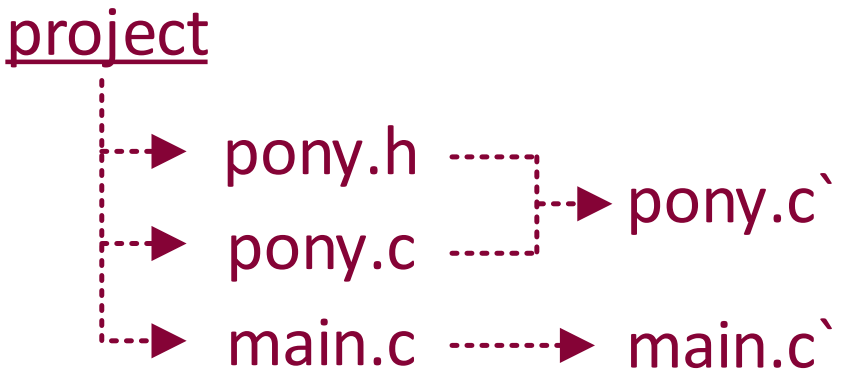


ПРЕПРОЦЕССИНГ

Операция осуществляется текстовым препроцессором.

Исходный код программы подвергается следующей обработке:

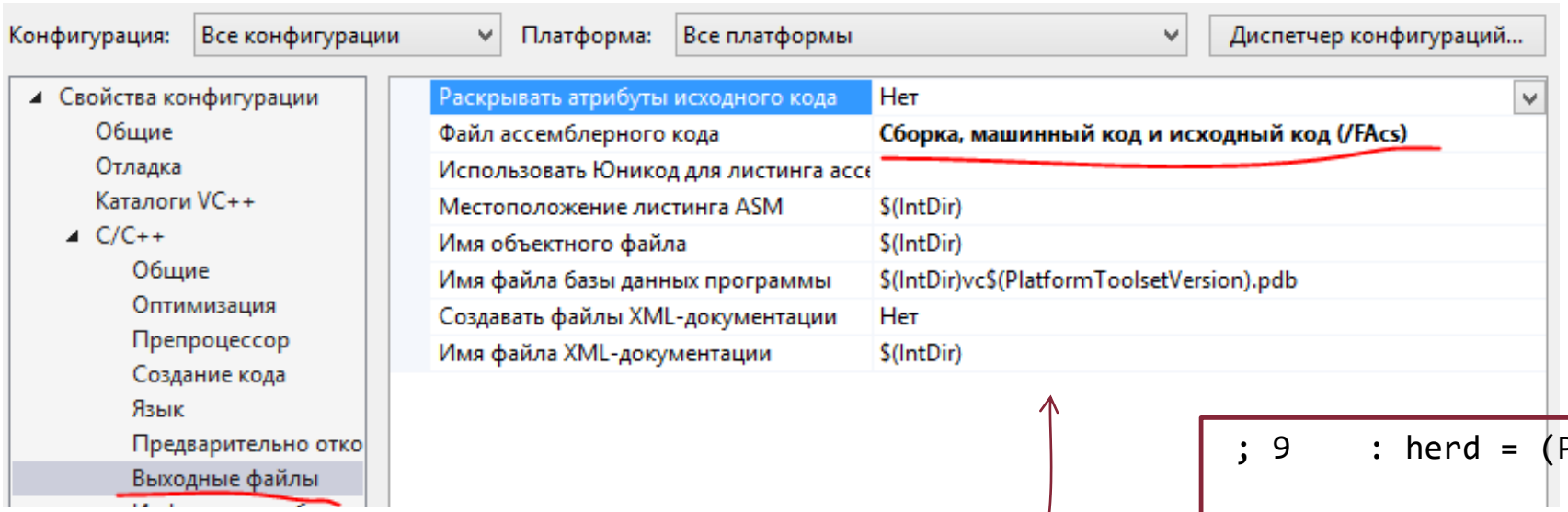
- Комментарии заменяются пустыми строками
- Текстовое включение файлов (обработка директивы `#include`)
- Выполнение макроподстановки (обработка директивы `#define`)
- Обрабатывается условная компиляция (обработка директив: `#if`, `#ifdef`, `#elif`, `#else`, `#endif`,)

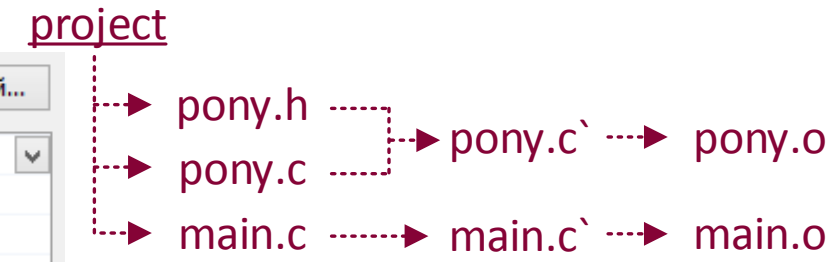


КОМПИЛЯЦИЯ

Преобразование полученного на этапе препроцессинга в ассемблерный код.

Машинный код – промежуточный шаг, между высокоуровневым языком и машинным (бинарным) кодом.





Для создания ассемблерного кода,
в Visual Studio можно задать настройку

Пример (calloc на ассемблере):

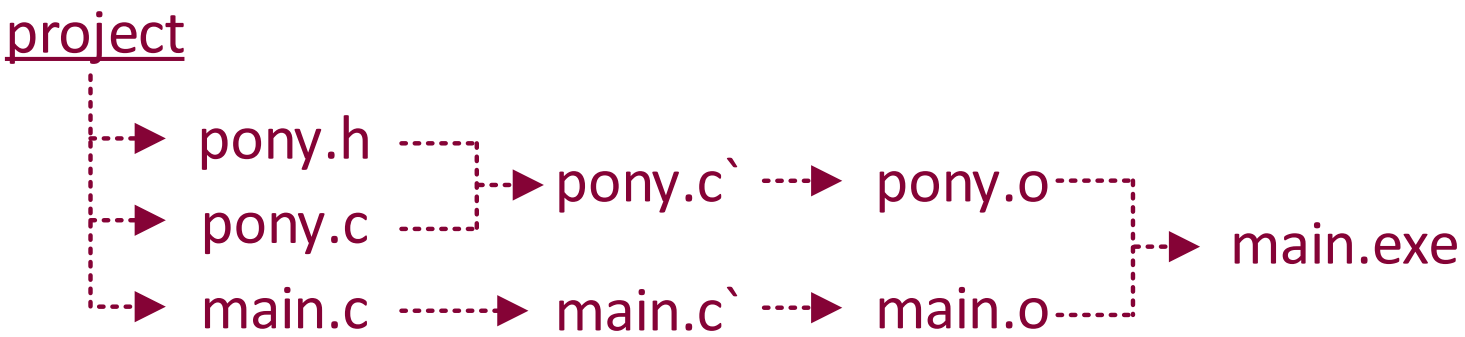
```
; 9    : herd = (PONY**)calloc(COUNT, sizeof(PONY*));

  00040ba 08 00 00 00 mov edx, 8
  00045b9 05 00 00 00 mov ecx, 5
  0004ae8 00 00 00 00 call calloc
  0004f48 98 cdqe
  0005148 89 45 08 mov QWORD PTR herd$[rbp], rax
```

ЛИНКОВКА

Данный этап связывает воедино все объектные файлы проекта.

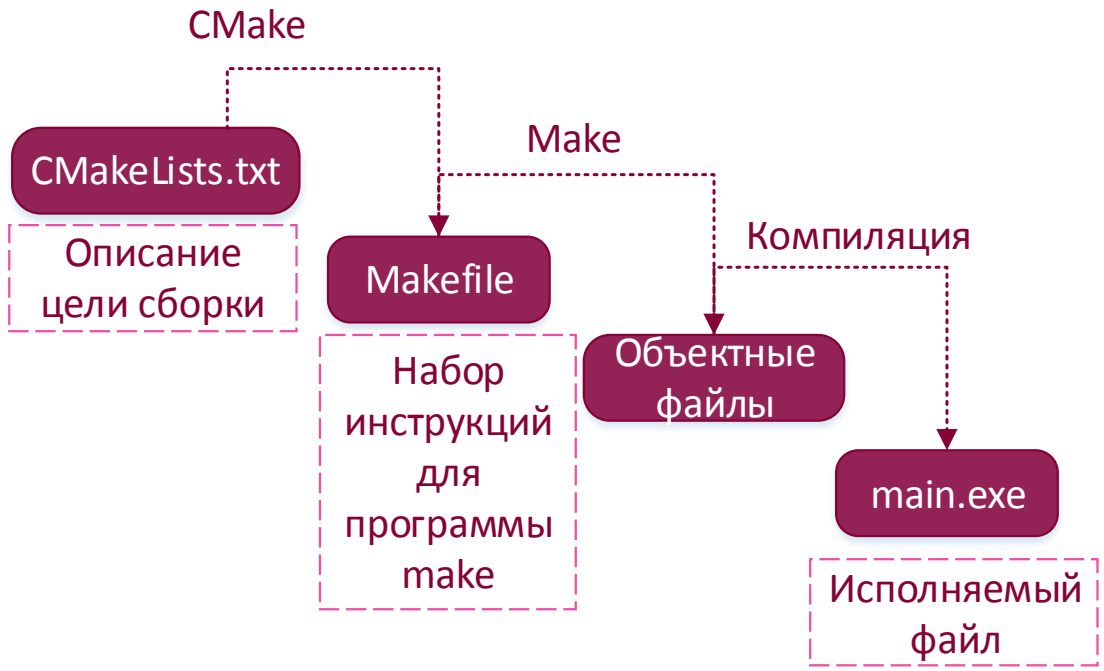
Также добавляет подключение библиотек, при необходимости.



CMAKE

Кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода.

Непосредственно сборкой не занимается, но подготавливает исходные файлы на языке высокого уровня к процессу компиляции.



CMAKELISTS.TXT

```
project(PONY)
cmake_minimum_required(VERSION 3.0)

find_package(*имя библиотеки* REQUIRED)

set(HEADER ${PROJECT_SOURCE_DIR}/header/*.h)
set(SOURCE ${PROJECT_SOURCE_DIR}/source/*.c)

add_executable(${CMAKE_PROJECT_NAME} ${SOURCE})

set(CMAKE_CXX_FLAGS "-Wall -Werror -Wextra -pedantic -g3 -Og")

message("Start cmake build!")

target_include_directories(${CMAKE_PROJECT_NAME} PUBLIC ${HEADER}))
target_link_libraries(${CMAKE_PROJECT_NAME} *имя библиотеки*)
```