Язык программирования C++
Многофайловая компоновка

Лекции по курсу

Языки программирования

семестр 3, ФИИТ

Михалкович С.С.

Пример 1

a.cpp

```
int main()
{
    i = 5;
    f();
}
```

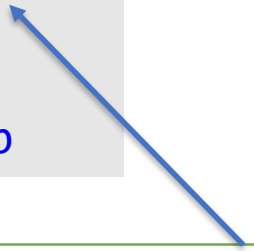
Ошибка. Неизвестные имена i и f

lib.cpp

```
#include <iostream>

int i;
void f()
{
    std::cout << i;
}
```

my.cppproj

```
a.cpp
lib.cpp
```



Проект объединяет несколько cpp-файлов

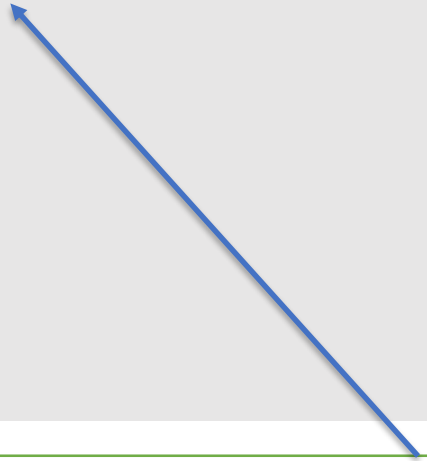
Пример 1

a.cpp

```
extern int i;
void f();

int main()
{
    i = 5;
    f();
}
```

lib.cpp

```
#include <iostream>

int i;
void f()
{
    std::cout << i;
}
```



OK

Объявления переменной i и функции f (не путать с определением!).

А если в проекте еще несколько cpp-файлов, использующих эти имена?

Необходимость заголовочных файлов

a.cpp

```
#include "lib.h"

int main()
{
    i = 5;
    f();
}
```

lib.cpp

```
#include <iostream>

int i;
void f()
{
    std::cout << i;
}
```

lib.h

```
extern int i;
void f();
```

Объявления выносим в **заголовочный файл**. И подключаем его к cpp-файлу с помощью директивы **препроцессора** `#include`

Заголовочный файл содержит лишь заголовки функций и объявления переменных

Необходимость заголовочных файлов

a.cpp

```
#include "lib.h"

int main()
{
    i = 5;
    f();
}
```

lib.cpp

```
#include <iostream>
#include "lib.h"

void f()
{
    std::cout << i;
}
int i;
```

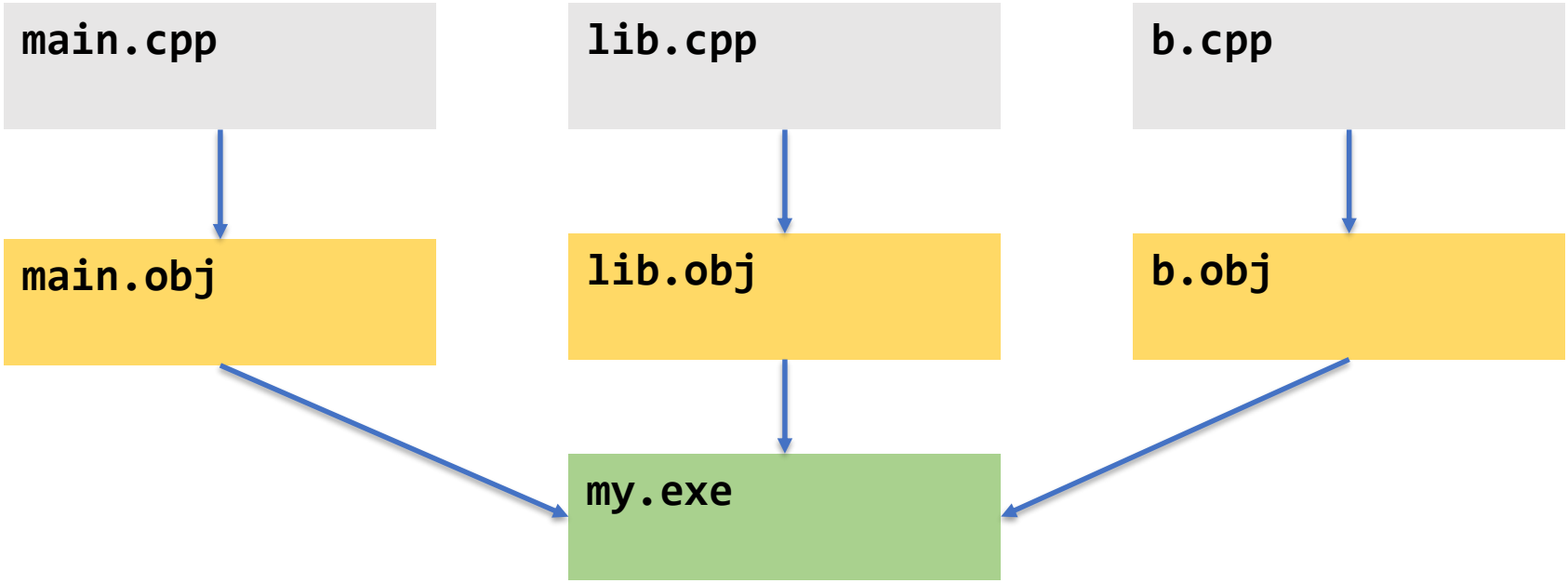
Принято также подключать заголовочный файл к "своему" cpp-файлу чтобы определения в нем можно было давать в любом порядке

lib.h

```
extern int i;
void f();
```

Независимая компиляция в C++

Все файлы проекта считаются равноценными. Компиляция начинается с любого файла



Этап компиляции – получение объектных файлов
Этап линковки (компоновки) – получение исполняемого файла из объектных

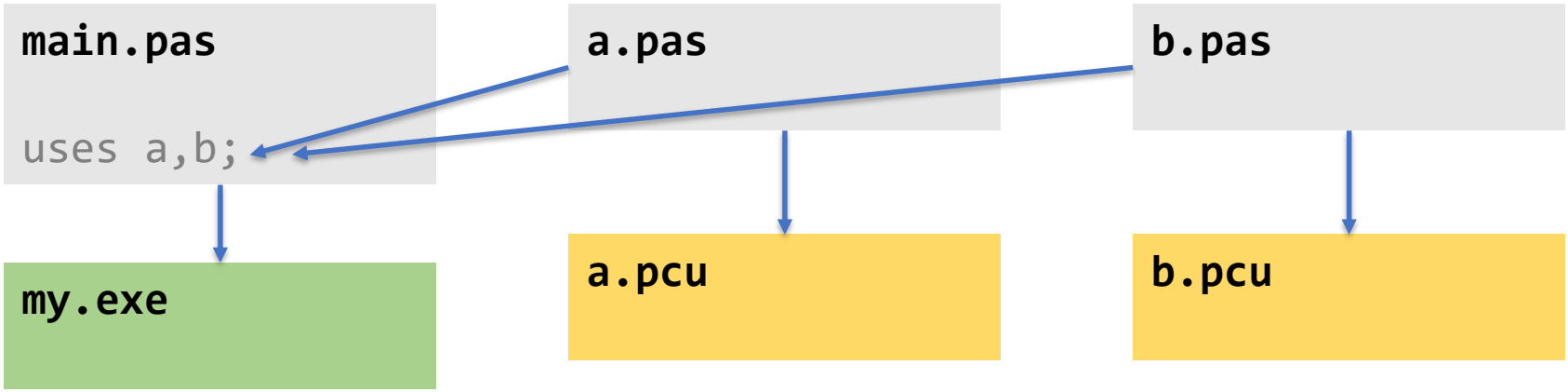
Линковщик (компоновщик)

Линковщик – программа, осуществляющая второй этап компиляции.
В каждом obj-файле указаны внешние ссылки (на объявленные внешние переменные и функции). Линковщик ищет их в остальных obj-файлах. Это долго, поэтому этап линковки долгий.

Компиляция, при которой каждый файл на первом этапе обрабатывается независимо от другого, называется **независимой**.

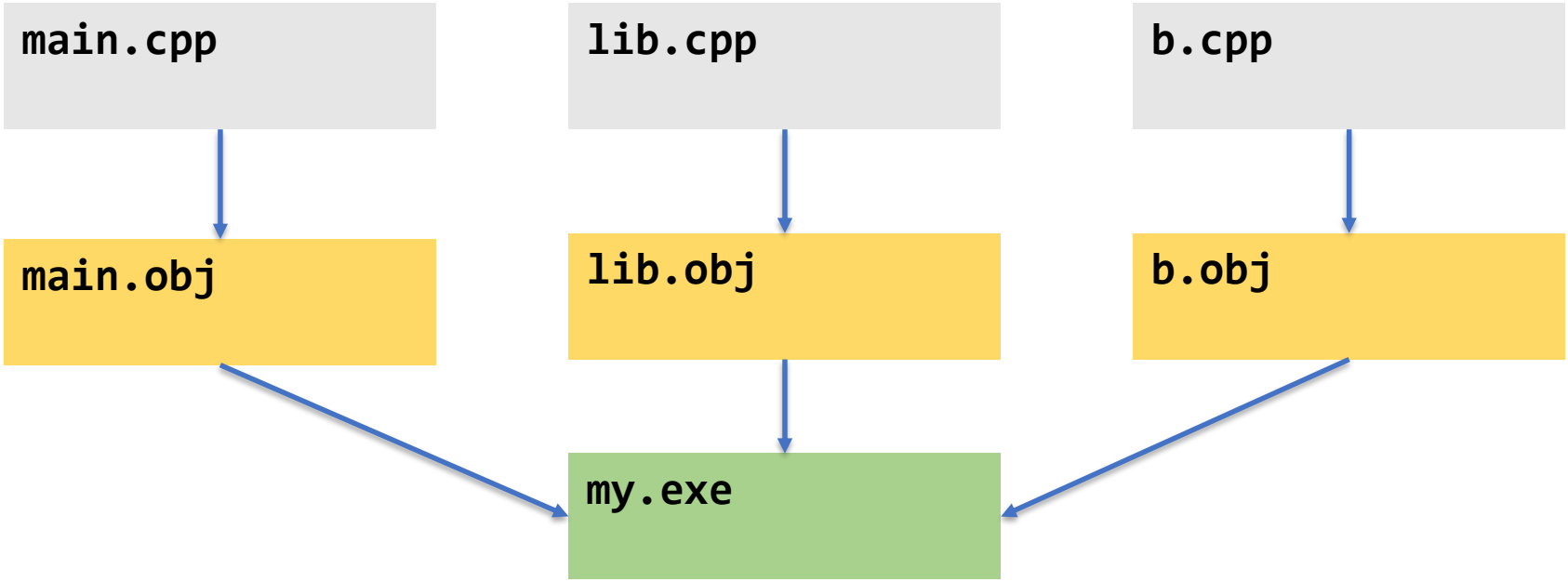
Раздельная компиляция в Pascal

Есть основная программа, к которой подключаются модули. Перед компиляцией основной программы все используемые в ней модули компилируются



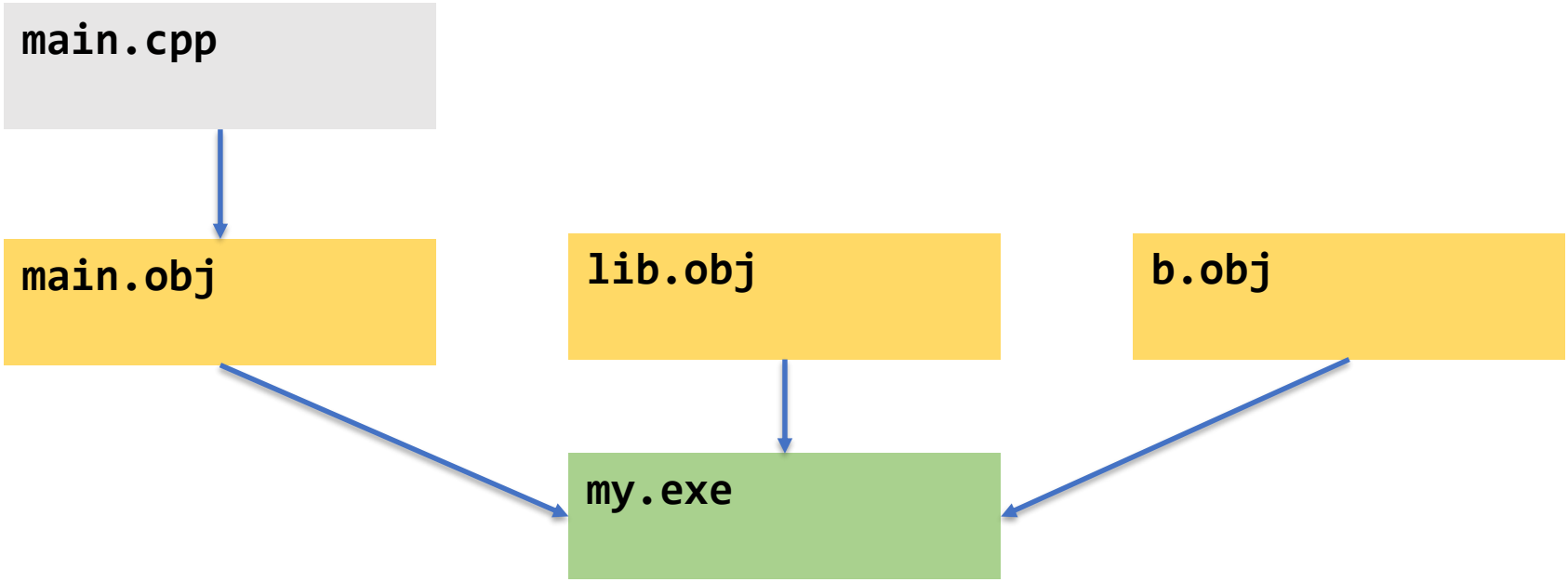
В Pascal необходимость в файле проекта отпадает так как есть основная программа – корень дерева компиляции
В Pascal pcu-файлы зависят от конкретного компилятора и используются чтобы повторно не перекомпилировать модуль если он не менялся

Независимая компиляция в C++



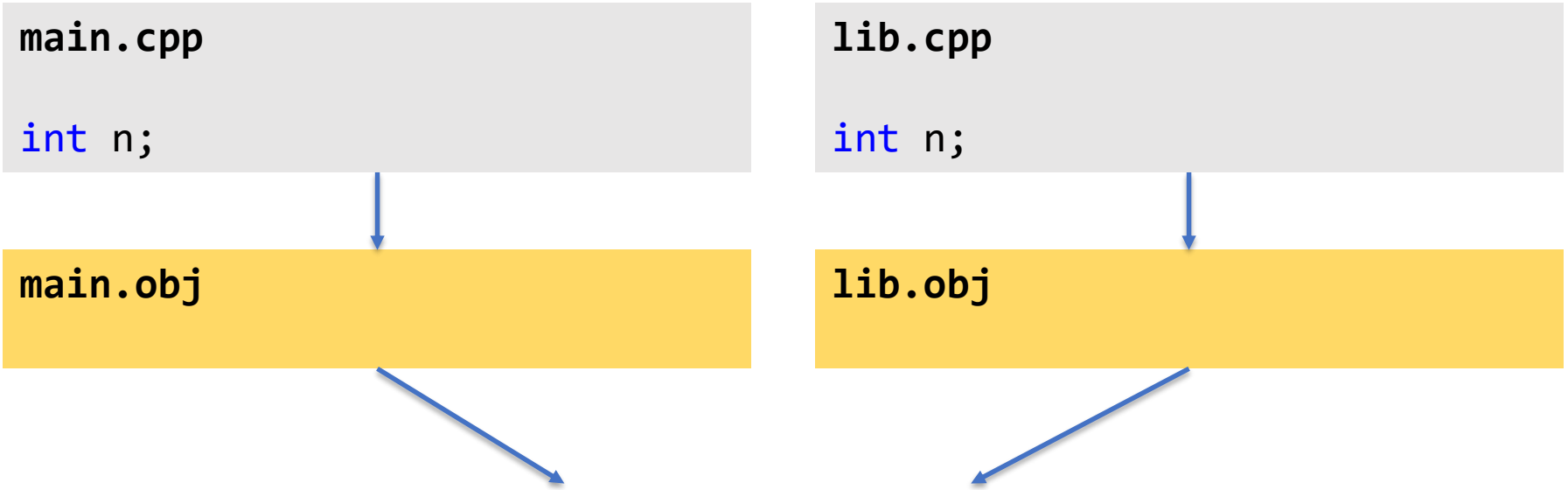
В C++ obj-файлы не перекомпилируются если соответствующий cpp-файл не менялся (по дате)

Независимая компиляция в C++



cpp-файлы могут вообще отсутствовать, тогда в проект добавляются соответствующие obj-файлы. Это используется для защиты интеллектуальной собственности

Глобальные описания в C++ и необходимость пространств имен



Ошибка линковки – одна переменная описана дважды
в одном и том же **глобальном безымянном пространстве имен**

Глобальные описания в Pascal

main.pas

```
uses lib;

var n: integer;

begin
  n := 2;
  m := 3;
end.
```

lib.pas

```
unit lib;

var n,m: integer;
```

Переменная n берется из основной программы т.к. разные переменные n принадлежат к разным пространствам имен

В Pascal не будет ошибки компиляции, т.к. каждый модуль образует неявное пространство имен. Его имя совпадает с именем модуля

Необходимость пространств имен в C++

main.cpp

```
int n; // глобальное ПИ

int main()
{
    n = 3;
    mylib::n = 2; // ошибка!
}
```

lib.cpp

```
namespace mylib
{
    int n;
}
```

Пространства имен в C++

main.cpp

```
namespace mylib
{
    extern int n;
}

int n;

int main()
{
    n = 3;
    mylib::n = 2; // OK
}
```

lib.cpp

```
namespace mylib
{
    int n;
}
```

Пространства имен и заголовочные файлы

main.cpp

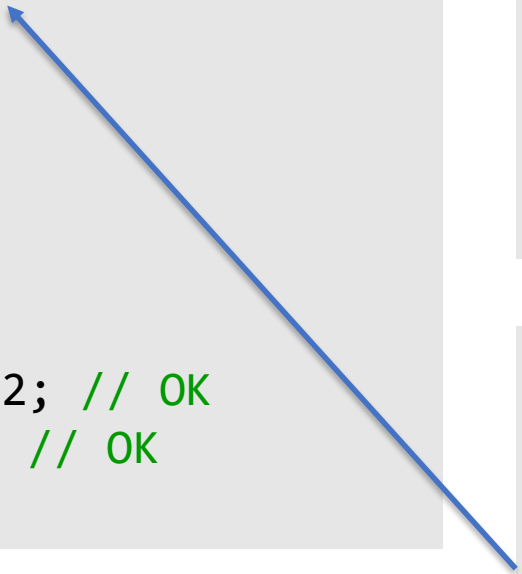
```
#include "lib.h"

int n;

int main()
{
    n = 3;
    mylib::n = 2; // OK
    mylib::f(); // OK
}
```

lib.cpp

```
namespace mylib
{
    int n,m;
    void f() {}
}
```

lib.h

```
namespace mylib
{
    extern int n;
    extern int m;
    void f();
}
```



Пространства имен и заголовочные файлы

main.cpp

```
#include "lib.h"

int n;

using namespace mylib;

int main()
{
    f(); // OK
    mylib::n = 2; // OK
    n = 3; // неоднозначность
}
```

lib.cpp

```
namespace mylib
{
    int n,m;
    void f() {}
}
```

lib.h

```
namespace mylib
{
    extern int n;
    extern int m;
    void f();
}
```

Пространства имен и заголовочные файлы

main.cpp

```
#include "lib.h"

int n;

using namespace mylib;

int main()
{
    f(); // OK
    mylib::n = 2; // OK
    ::n = 3;
}
```

lib.cpp

```
namespace mylib
{
    int n,m;
    void f() {}
}
```

lib.h

```
namespace mylib
{
    extern int n;
    extern int m;
    void f();
}
```

Пространства имен и заголовочные файлы

main.cpp

```
#include "lib.h"

int n;

using mylib::m;

int main()
{
    f(); // OK
    n = 2; // OK
    m = 3; // OK
}
```

lib.cpp

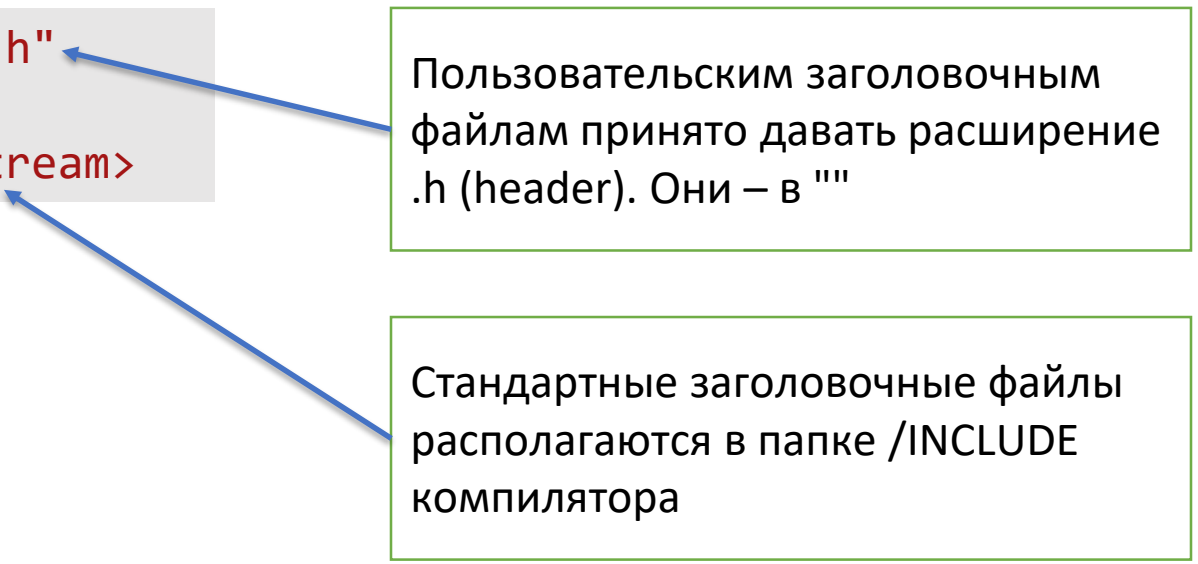
```
namespace mylib
{
    int n,m;
    void f() {}
}
```

lib.h

```
namespace mylib
{
    extern int n;
    extern int m;
    void f();
}
```

Стандартные и пользовательские заголовочные файлы

```
#include "lib.h"

#include <iostream>
```



Где находится код стандартных подпрограмм?

Он находится в ряде стандартных .obj-файлах, которые неявно подключаются к любому проекту.
Обычно стандартные .obj-файлы объединяются в .lib-файл и именно он неявно подключается к проекту

Q & A