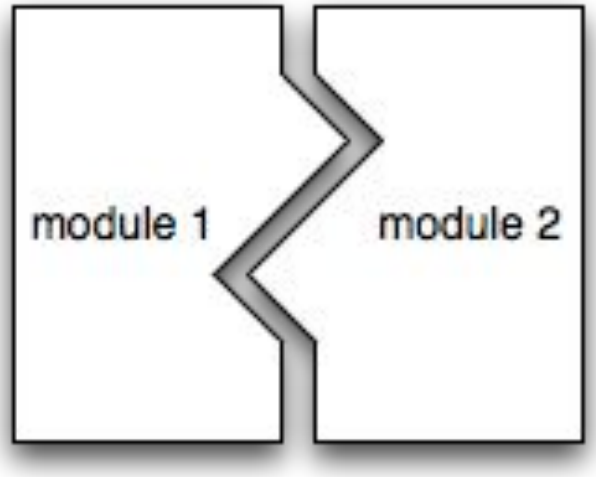
Модульное программирование



Разбиение программы на модули (файлы)

- Самые простые программы могут состоять из одной функции main
- Чуть более сложные включают в себя другие функции
- По мере возрастания сложности программы функций становится слишком много, в них становится тяжело ориентироваться
- Выход – разбиение функций на отдельные модули по смысловому значению

Проектирование «сверху вниз»
«top-down design»

Top-down design - последовательное понижение уровня детализации задачи

Задача. Найти медиану значений элементов массива

Медиана – полусумма срединных значений ранжированного ряда

возьмем ряд

23, 11, 21, 5, 17, 66, 34, 9

..и ранжируем (упорядочим) его:

5, 9, 11, 17, 21, 23, 34, 66

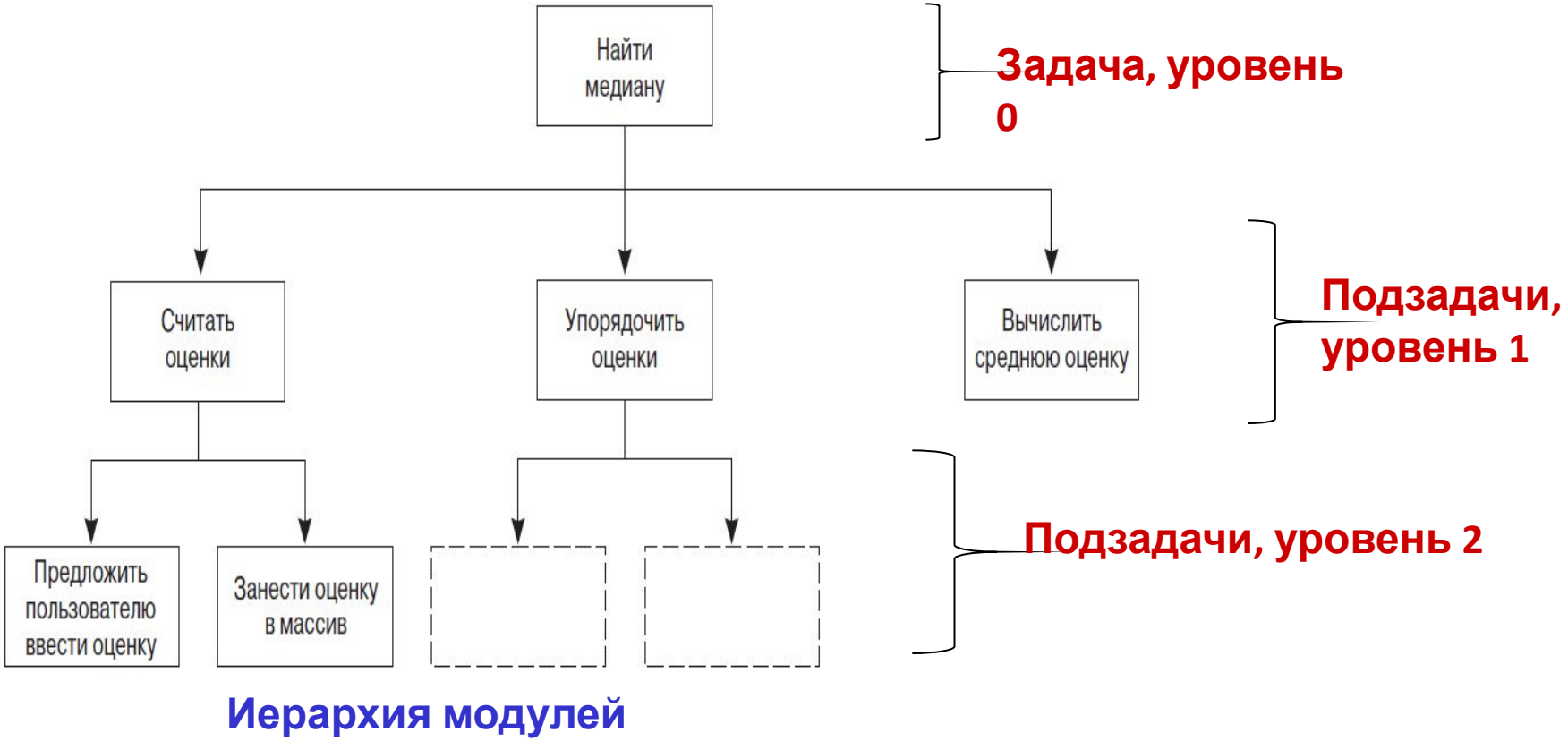
Срединными (лежащими в середине) значениями являются числа 17 и 21.

Их сумма: 17 + 21 = 38

Медиана данного ряда: 38 / 2 = 19

Используется для расчета, например, среднего значения зарплаты по всей компании, среднего срока нахождения товаров на складе, среднего количества участников форумов.

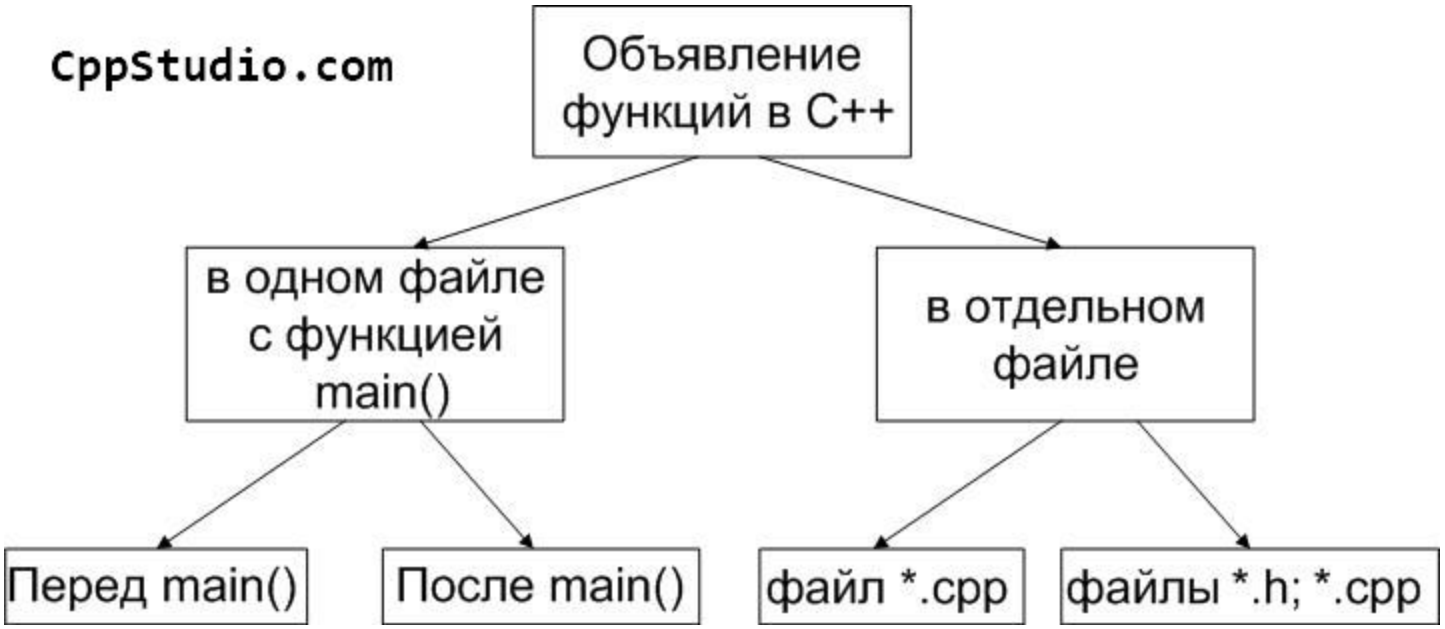
Проектирование «сверху вниз»
«top-down design»



Иерархия модулей

(см. Example4, Project Median)

Использование функций



(см. Example4, Project MedianSecond, MedianThird)

Реализация функций в отдельном файле

Файл add.cpp

```
int add(int x, int y)
{
    return x + y;
}
```

Файл main.cpp

```
#include <iostream>
int main()
{
    using namespace std;
    cout<<"The sum of 3 and 4 is: "<<add(3, 4)<<endl;
    return 0;
}
```

Compile-time error

add.cpp(10) : error C3861: 'add': identifier not found

Реализация функций в отдельном файле

Файл add.cpp

```
int add(int x, int y)
{
    return x + y;
}
```

Файл main.cpp

```
#include <iostream>
int add(int x, int y); // прототип функции
int main()
{
    using namespace std;
    cout<<"The sum of 3 and 4 is: "<<add(3, 4)<<endl;
    return 0;
}
```

Как организуется модуль на языке C++?

- Модуль логически состоит из двух файлов - файла с исходным кодом (source file) и заголовочного файла (header file)
  - файл с исходным кодом (module.cpp) включает в себя определения функций, а также определения глобальных переменных и констант (если они есть); в первой строке обычно подключается заголовочный файл того же модуля:

```
#include "module.h"
```

  - заголовочный файл (module.h) включает в себя прототипы функций, определения констант, объявления глобальных переменных - но только для тех элементов модуля, о которых должны знать другие модули

Как организуется модуль на языке C++?

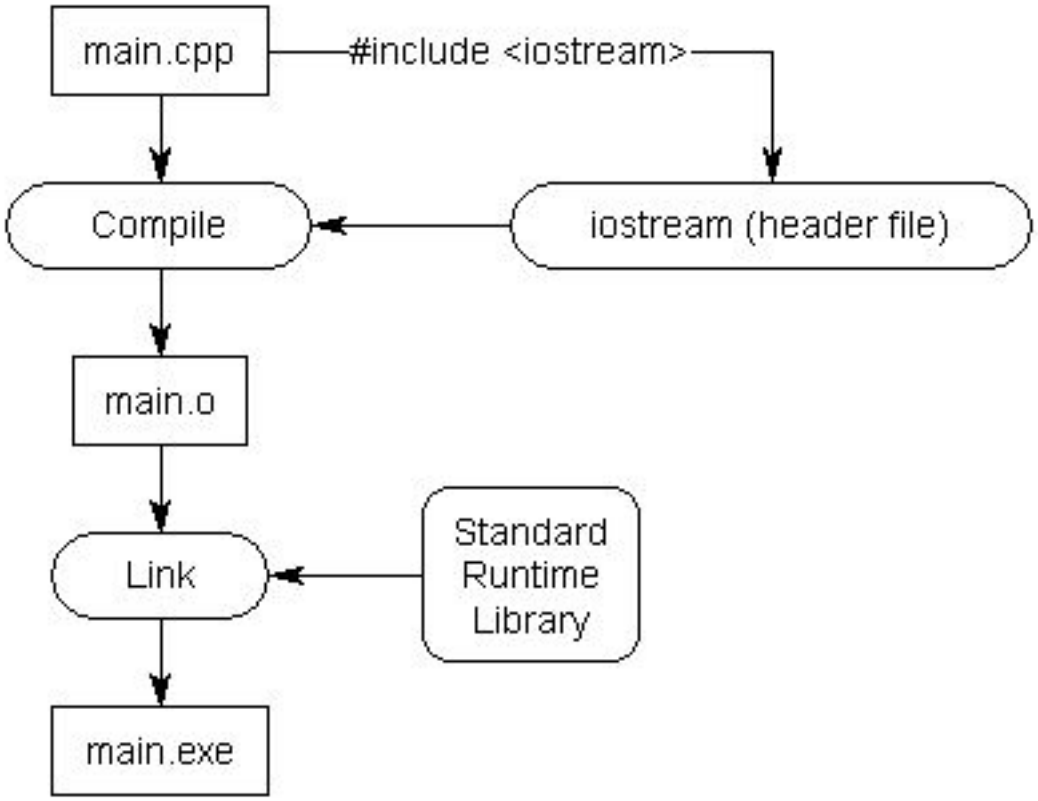
- Если какой-либо модуль использует данный, необходимо подключить его заголовочный файл (в двойных кавычках – это указывает на то, что заголовочный файл не системный, а собственный):

```
#include "module.h"
```

- Главный модуль программы обычно содержит только функцию main и не имеет заголовочного файла (поскольку функция main не используется в других модулях)
- Некоторые модули включают только заголовочный файл – например, содержащий определения глобальных констант

Заголовочные файлы

```
#include <iostream>
int main()
{
    using namespace std;
    cout<<"Hello, world!"<<endl;
    return 0;
}
```



Как происходит сборка программы?

- Сборка (build) состоит из 3 основных этапов:
  - на этапе препроцессинга (preprocessing) директивы препроцессора (например, #include) заменяются содержимым указанного в них заголовочного файла, в результате файл с исходным кодом дополняется прототипами указанных там функций и объявлениями глобальных переменных – за счет этого в файле можно вызывать указанные функции (прототип впереди) и использовать глобальные переменные (объявление впереди). Препроцессор может создавать на диске временные файлы, которые, однако, удаляются после окончания сборки

Как происходит сборка программы?

- Сборка (build) состоит из 3 основных этапов:
  - на этапе компиляции (compiling) для каждого исходного файла составляются таблицы определенных в нем функций и глобальных переменных, все определенные функции переводятся на машинный язык (при переводе на машинный язык могут выявляться ошибки компиляции). Результатом работы компилятора являются файлы module.obj (объектные файлы) для каждого использованного в программе модуля

Как происходит сборка программы?

- Сборка (build) состоит из 3 основных этапов:
  - на этапе связывания (linking) происходит привязка всех используемых (вызванных) функций и глобальных переменных к той таблице, в которой они определены; если определения не обнаружены ни в одной таблице, происходит ошибка связывания (unresolved external symbol …). Результатом связывания является файл program.exe (для MVS имя совпадает с именем проекта) – исполняемый файл

Что при сборке происходит с библиотечными функциями?

- Прототипы функций находятся в заголовочных файлах (iostream, math.h, stdio.h, string.h и т.п.)
- Объектные файлы с уже переведенными на машинный язык определениями функций заранее собраны в библиотеки (файлы с расширением lib или dll)
  - определения из статических библиотек (расширение lib) на этапе связывания добавляются в исполняемый файл программы
  - определения из динамических библиотек (расширение dll) в исполняемый файл не добавляются; вместо этого в ссылке на соответствующую функцию указывается, что она находится в динамической библиотеке, и при ее вызове происходит обращение к библиотеке
  - поэтому, статические библиотеки нужны только на этапе связывания, а динамические – и на этапе исполнения программы

Заголовочные файлы

Файл add.h

```
#ifndef ADD_H
#define ADD_H

int add(int x, int y);

#endif
```

Файл add.cpp

```
int add(int x, int y)
{
    return x + y;
}
```

Файл main.cpp

```
#include <iostream>
#include “add.h”

int main()
{
    using namespace std;
    cout<<"The sum of 3 and 4 is: "<<add(3, 4)<<endl;
    return 0;
}
```

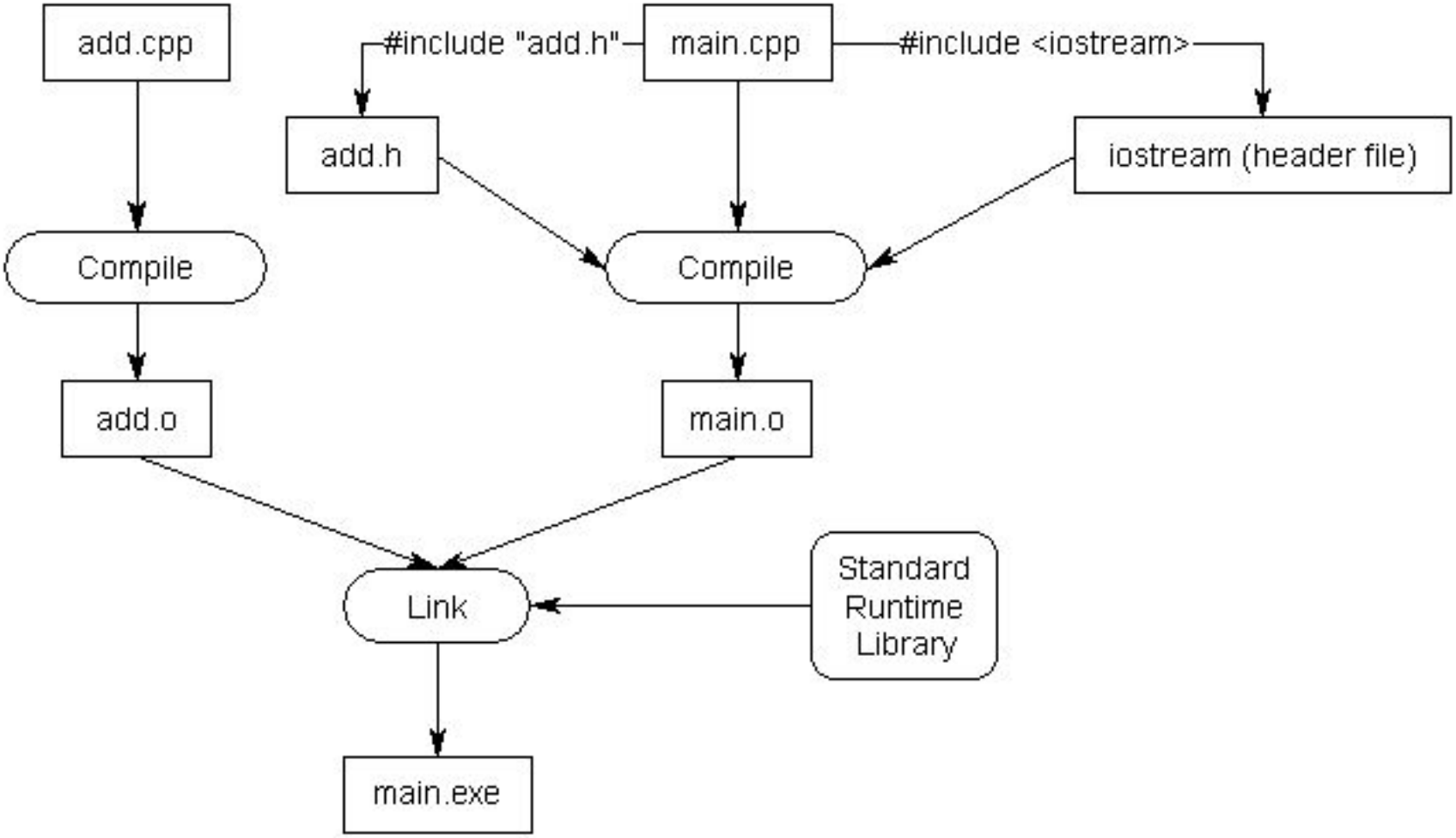
Что такое #ifndef-#define-#endif?

- Это, как и #include, директивы препроцессора
- Директива #define позволяет определить переменную препроцессора (например, ADD_H)
- Директива #ifndef позволяет выяснить, определена ли переменная препроцессора. Если она не определена, участок кода до директивы

#endif вставляется в программу, если же определена – выбрасывается из нее

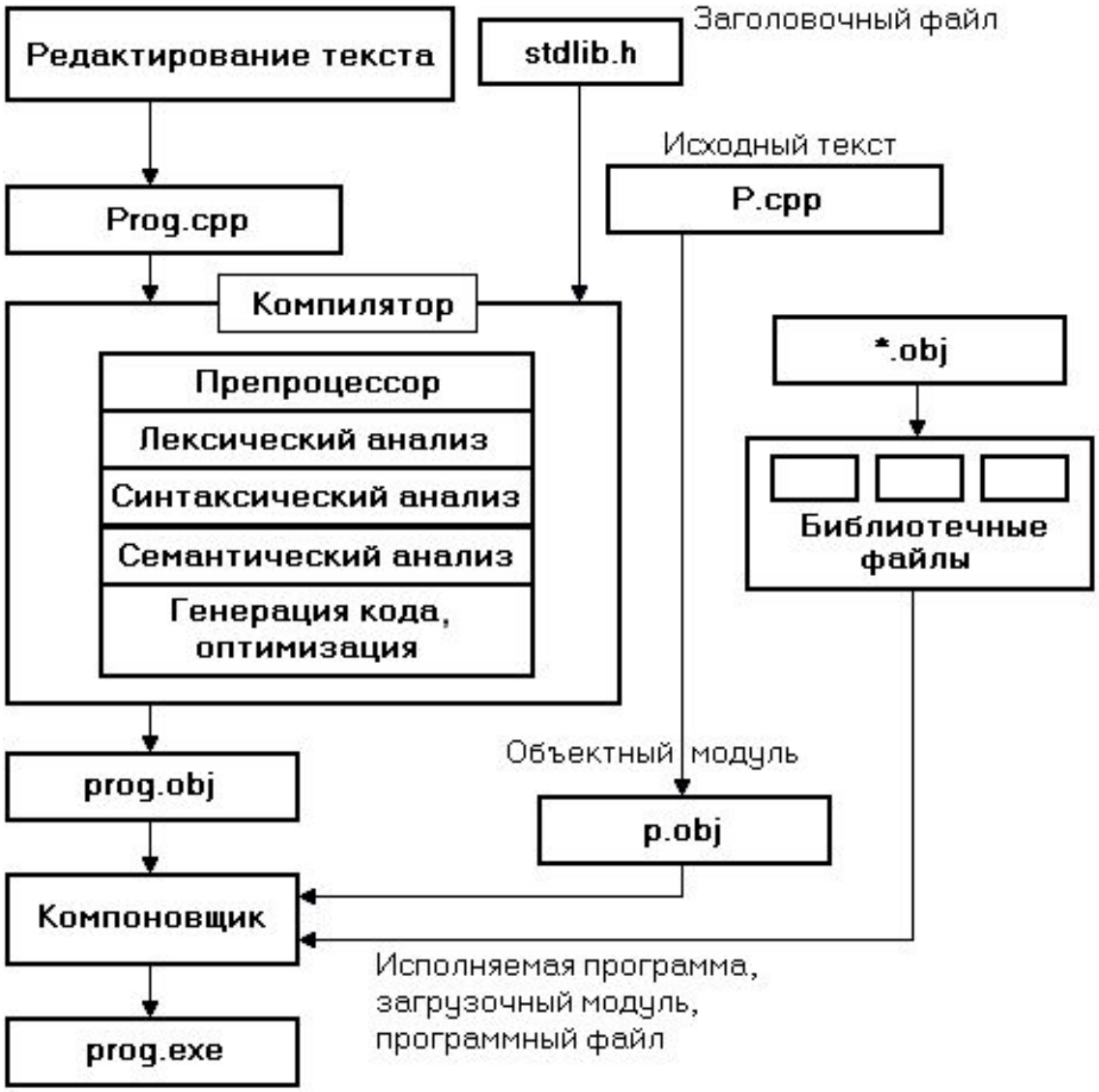
- Нужны эти скобки для того, чтобы текст заголовочного файла не мог вставиться в исходный файл дважды

Заголовочные файлы



(см. Example4, UsingHeader)

Компиляция и компоновка



Правила организации header-файлов

1. Не объявляйте переменные в header-файлах, за исключением случаев когда вы объявляете константу

2. Не включайте реализацию функций (за исключением тривиальных функций)

3. Каждый header-файл должен быть независимым насколько это возможно (все объявления для функциональности модуля A поместите в файл A.h, все объявления для функциональности модуля B поместите в файл B.h)

4. Старайтесь включать в header-файл другие header-файлы насколько это возможно