# Программирование на языке C++
# Лекция 11

## Сборка программы

# Исходный код

# Программа

```
#include <iostream>

int main(){
    std::cout << "Hello, World!";
    return 0;
}
```

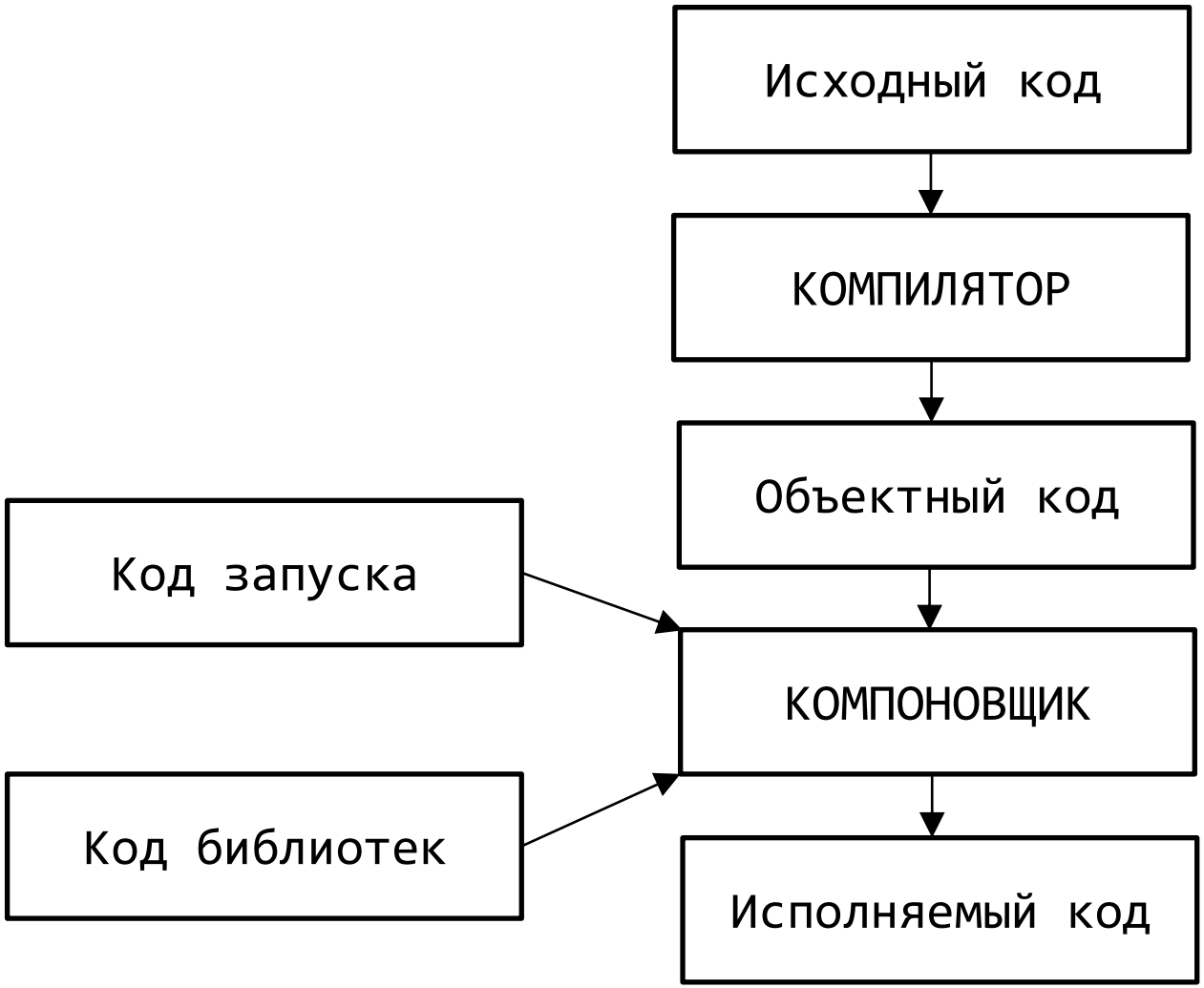
# Форматы файла с исходным кодом

| Реализация C++ | Расширения файла исходного кода |
|---|---|
| Unix | C, cc, cxx, c |
| GNU C++ | C, cc, cxx, cpp, c++ |
| Digital Mars | cpp, cxx |
| Borland C++ | cpp |
| Watcom | cpp |
| Microsoft Visual C++ | cpp, cxx, cc |
| Freestyle CodeWarrior | cpp, cp, cc, cxx, c++ |

# Зачем разделять исходный код на части?

- Разделение на отдельные, слабо связанные части, чтобы не держать всю программу в голове;
- Совместная работа нескольких программистов;
- Для ускорения компиляции программы.

# Этапы сборки программы



# Препроцессинг I

Препроцессинг – подготовка исходного кода к процессу компиляции.

Основные действия препроцессора:

- Замена идентификаторов заранее подготовленными последовательностями символов;
  `#define ABC 100`
- Включение в программу текстов из указанных файлов;
  `#include <math.h>`

# Препроцессинг II

- исключение из программы отдельных частей ее текста, условная компиляция;
  #ifndef ABC
  истина, если идентификатор ABC не определен в настоящий момент.
  #else
  . . .
  #endif
- макроподстановка, то есть замена обозначения параметризованным текстом, формируемым препроцессором с учетом конкретных аргументов.
  #define abs(A) (((A) > 0)?(A) : -(A))

# Ассемблирование

Ассемблирование – преобразование (трансляция) текста исходного кода на языке C++ в ассемблерный код.

Ассемблирование не является обязательным процессом обработки файлов на языке C++. Некоторые компиляторы могут не выполнять данный шаг.

# Компиляция

Компиляция – процесс преобразования текста исходного кода на языке C++ напрямую или ассемблерного кода в объектный файл.

Объектный файл — это бинарный файл, то есть состоящий из конкретных инструкция для процессора. Фактически объектный файл - это набора функций.

Обычно расширение объектного файла .o

main.cpp -> main.o

# Программа

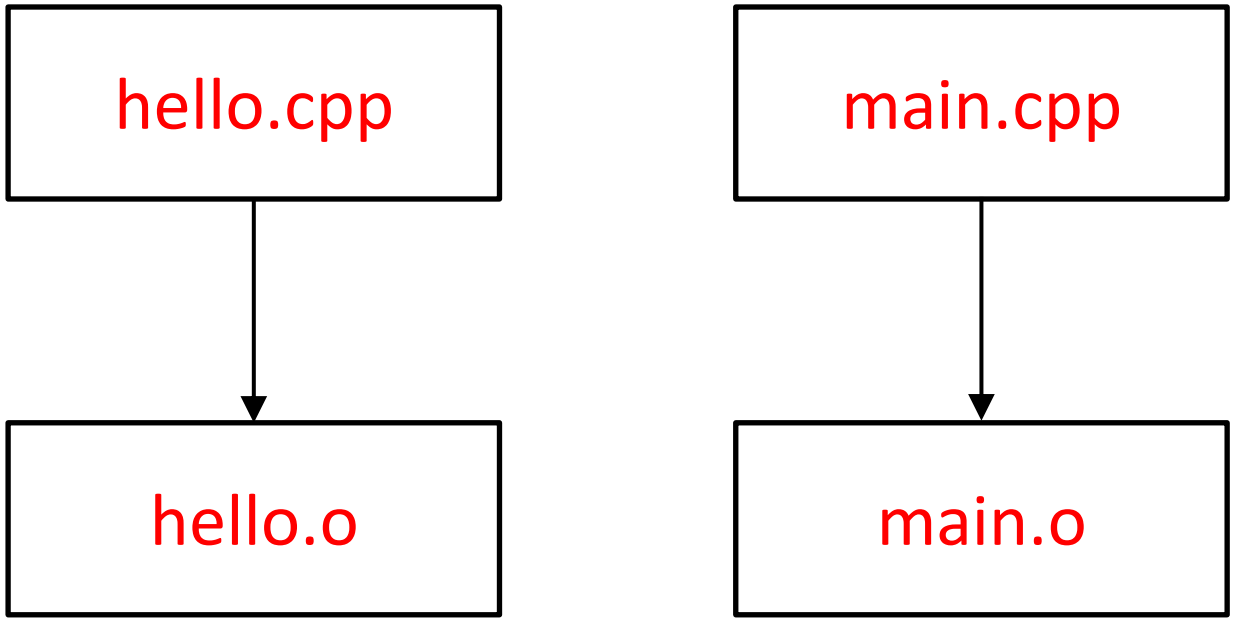
hello.cpp

```
#include <iostream>

void hello(){
    std::cout << "Hello, World!";
}
```

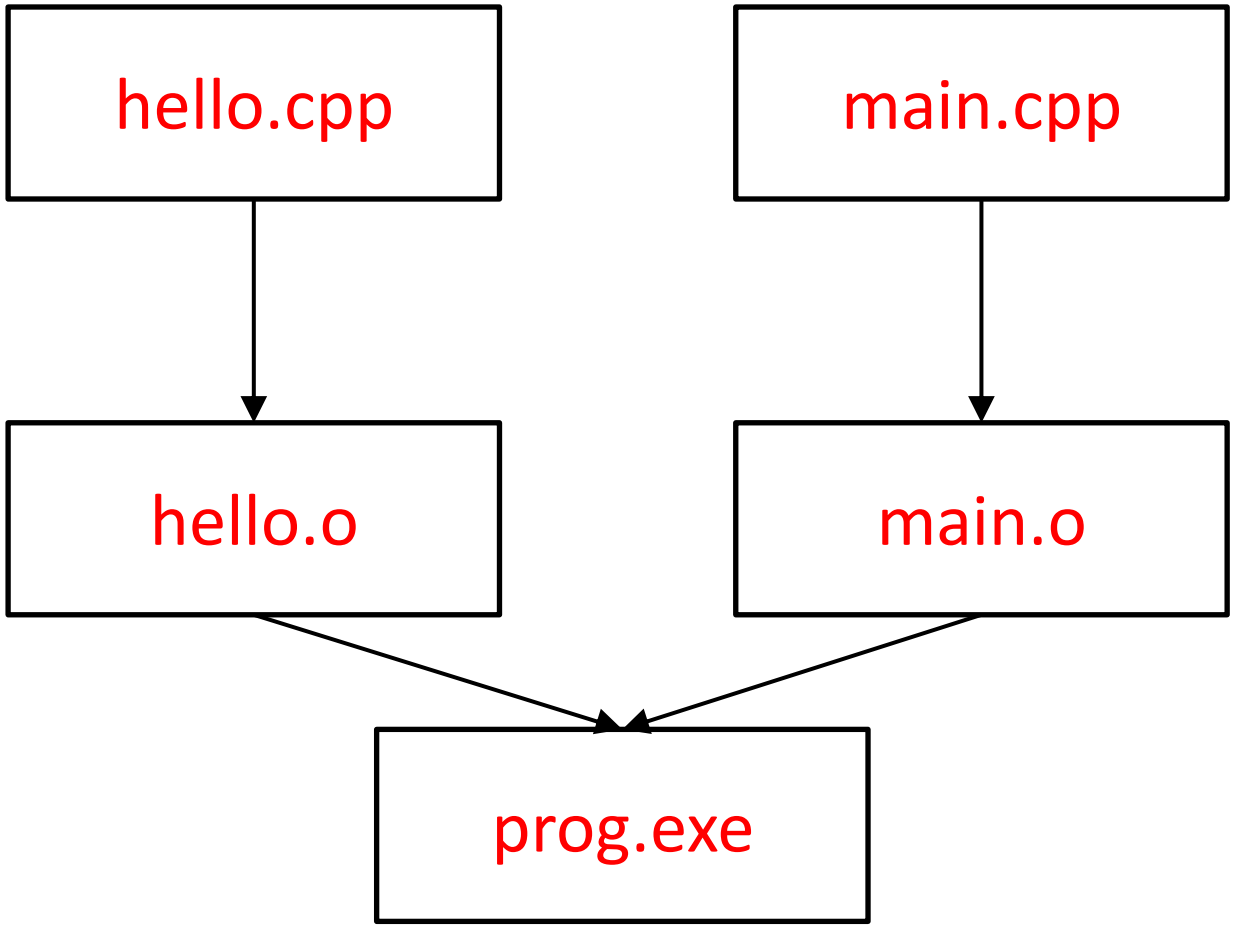
main.cpp

```
void hello();

int main(){
    hello();
    return 0;
}
```

# Компиляция



# Линковка (Компоновка)



# Компоновка (Линковка)

**Компоновщик (линкер)** связывает все объектные файлы и статические библиотеки в единый исполняемый файл, который мы и сможем запустить в дальнейшем.

# Модифицировали один из файлов

hello.cpp

```
#include <iostream>

void hello(int n){
    std::cout << n;
}
```

main.cpp

```
void hello();

int main(){
    hello();
    return 0;
}
```

# Вернёмся назад и добавим объявления

hello.cpp

```
#include <iostream>
void hello();

void hello(){
    std::cout << "Hello, World!";
}
```

main.cpp

```
void hello();

int main(){
    hello();
    return 0;
}
```

# Вынесем объявления в отдельный файл

hello.cpp

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

void hello(){
    std::cout << "Hello, World!";
}
```

hello.h

```
void hello();
```

main.cpp

```
#include "hello.h"

int main(){
    hello();
    return 0;
}
```

# Снова изменим код

hello.cpp

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

void hello(int n){
    std::cout << n;
}
```

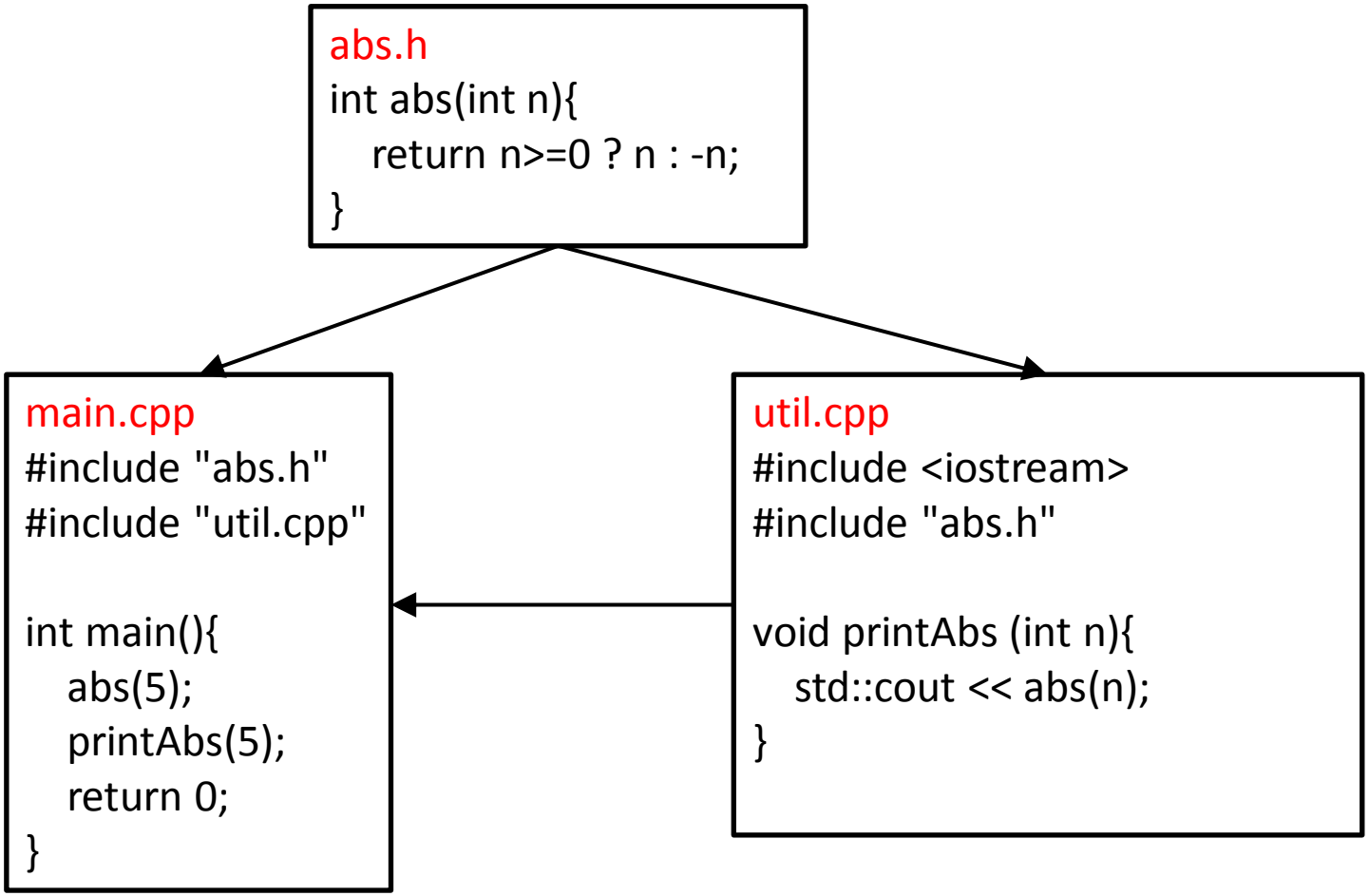
main.cpp

```
#include "hello.h"

int main(){
    hello();
    return 0;
}
```

hello.h

```
void hello();
```

# Проблемы заголовочных файлов I
# Определение функции в .h файле

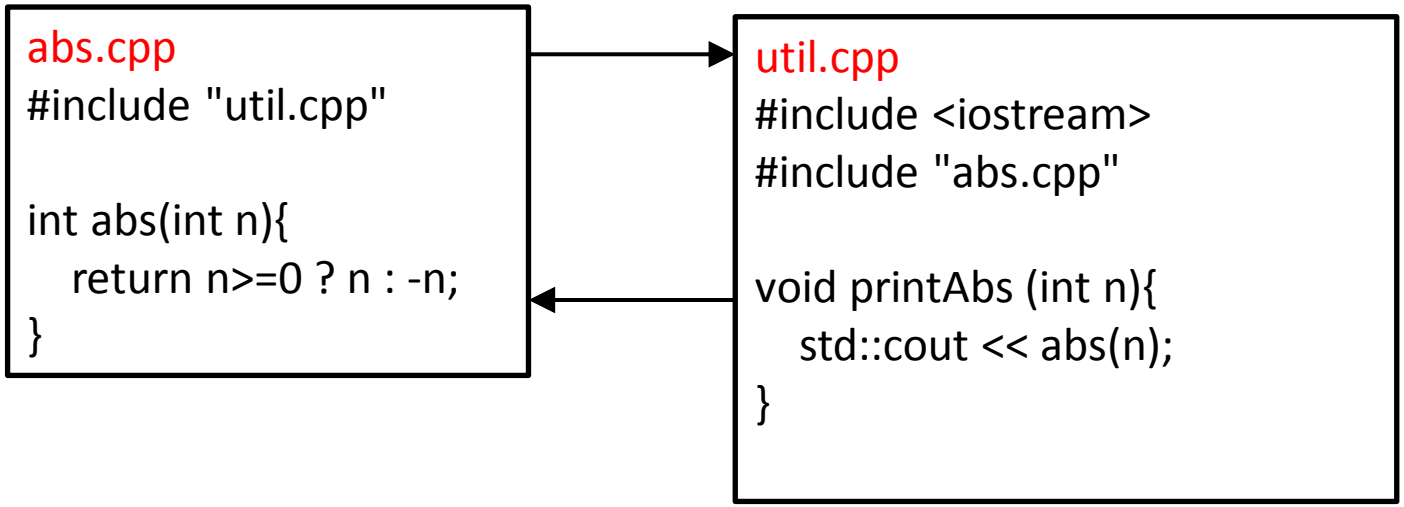


# Проблемы заголовочных файлов I
# Определение функции в .h файле

main.cpp

```
int abs(int n){
    return n>=0 ? n : -n;
}

#include <iostream>
int abs(int n){
    return n>=0 ? n : -n;
}

void printAbs (int n){
    std::cout << abs(n);
}

int main(){
    abs(5);
    printAbs(5);
    return 0;
}
```

# Проблемы заголовочных файлов II
# Перекрёстное включение



# Проблемы заголовочных файлов. Решение

```
// Нестандартная, но широко распространённая директива
#pragma once

// Обычный способ
#ifndef GRANDFATHER_H
#define GRANDFATHER_H
struct foo {
    int member;
};
#endif /* GRANDFATHER_H */
```