Лекция 2. Модульное программирование

Объектно-ориентированное программирование на C++

7 сентября 2009 г.

Проблемы при разработке ПО

Проблемы, возникающие при работе над крупными проектами

- Время компиляции.
- Организация коллективной работы над проектом.
- Грамотная организация исходного кода.
- Повторное использование собственного и чужого кода.

Технология модульного программирования включает

- поддержку со стороны языка;
- набор правил для разработчиков.

Проблемы при разработке ПО

Проблемы, возникающие при работе над крупными проектами

- Время компиляции.
- Организация коллективной работы над проектом.
- Грамотная организация исходного кода.
- Повторное использование собственного и чужого кода.

Технология модульного программирования включает

- поддержку со стороны языка;
- набор правил для разработчиков.

Проблемы при разработке ПО

Проблемы, возникающие при работе над крупными проектами

- Время компиляции.
- Организация коллективной работы над проектом.
- Грамотная организация исходного кода.
- Повторное использование собственного и чужого кода.

Технология модульного программирования включает

- поддержку со стороны языка;
- набор правил для разработчиков.

Проблемы при разработке ПО

Проблемы, возникающие при работе над крупными проектами

- Время компиляции.
- Организация коллективной работы над проектом.
- Грамотная организация исходного кода.
- Повторное использование собственного и чужого кода.

Технология модульного программирования включает

- поддержку со стороны языка;
- набор правил для разработчиков.

Проблемы при разработке ПО

Проблемы, возникающие при работе над крупными проектами

- Время компиляции.
- Организация коллективной работы над проектом.
- Грамотная организация исходного кода.
- Повторное использование собственного и чужого кода.

Технология модульного программирования включает

- поддержку со стороны языка;
- набор правил для разработчиков.

Проблемы при разработке ПО

Проблемы, возникающие при работе над крупными проектами

- Время компиляции.
- Организация коллективной работы над проектом.
- Грамотная организация исходного кода.
- Повторное использование собственного и чужого кода.

Технология модульного программирования включает

- поддержку со стороны языка;
- набор правил для разработчиков.

Модульный подход к проектированию ПО

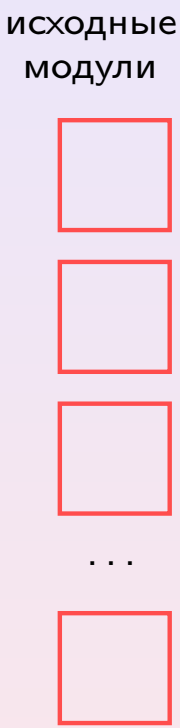

Рис. 1: Схема модульного подхода

Модульный подход к проектированию ПО

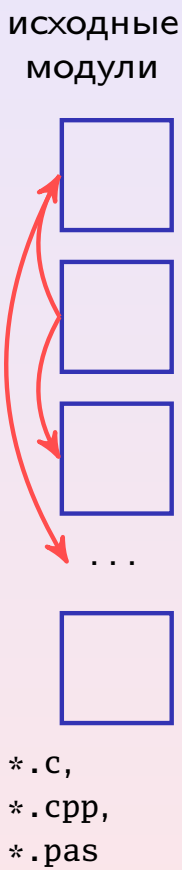

Рис. 1: Схема модульного подхода

Модульный подход к проектированию ПО

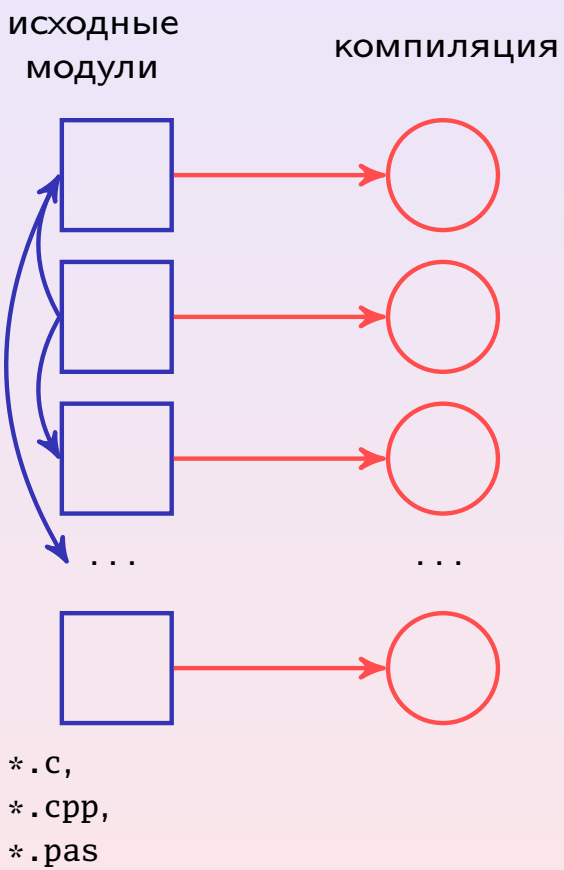

Рис. 1: Схема модульного подхода

Модульный подход к проектированию ПО

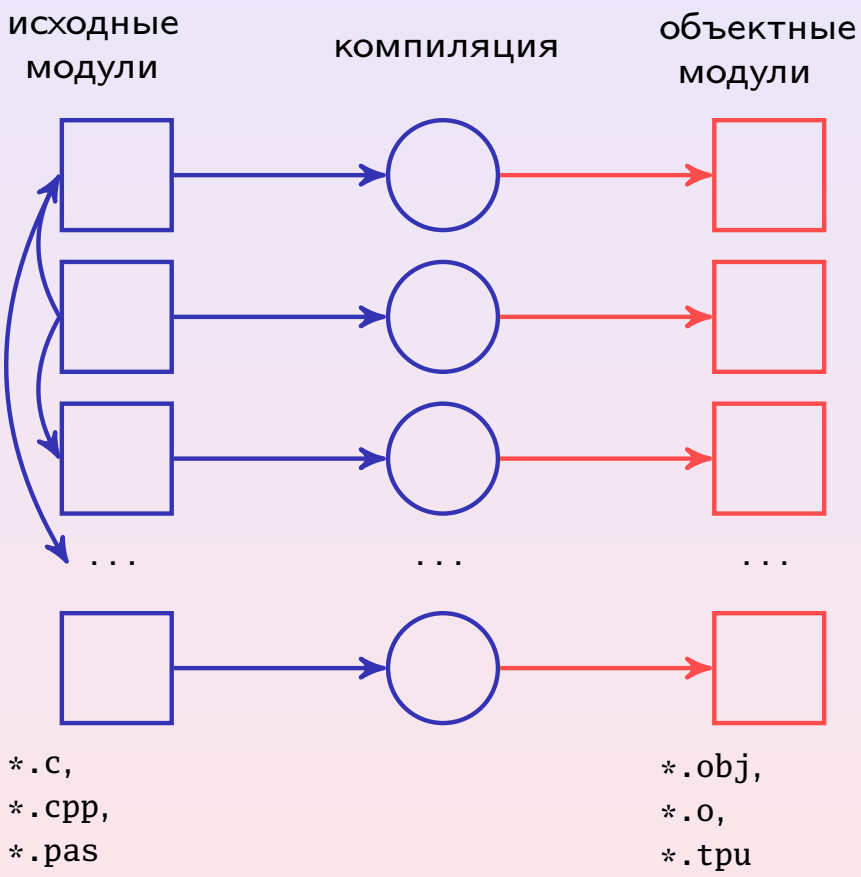

Рис. 1: Схема модульного подхода

Модульный подход к проектированию ПО

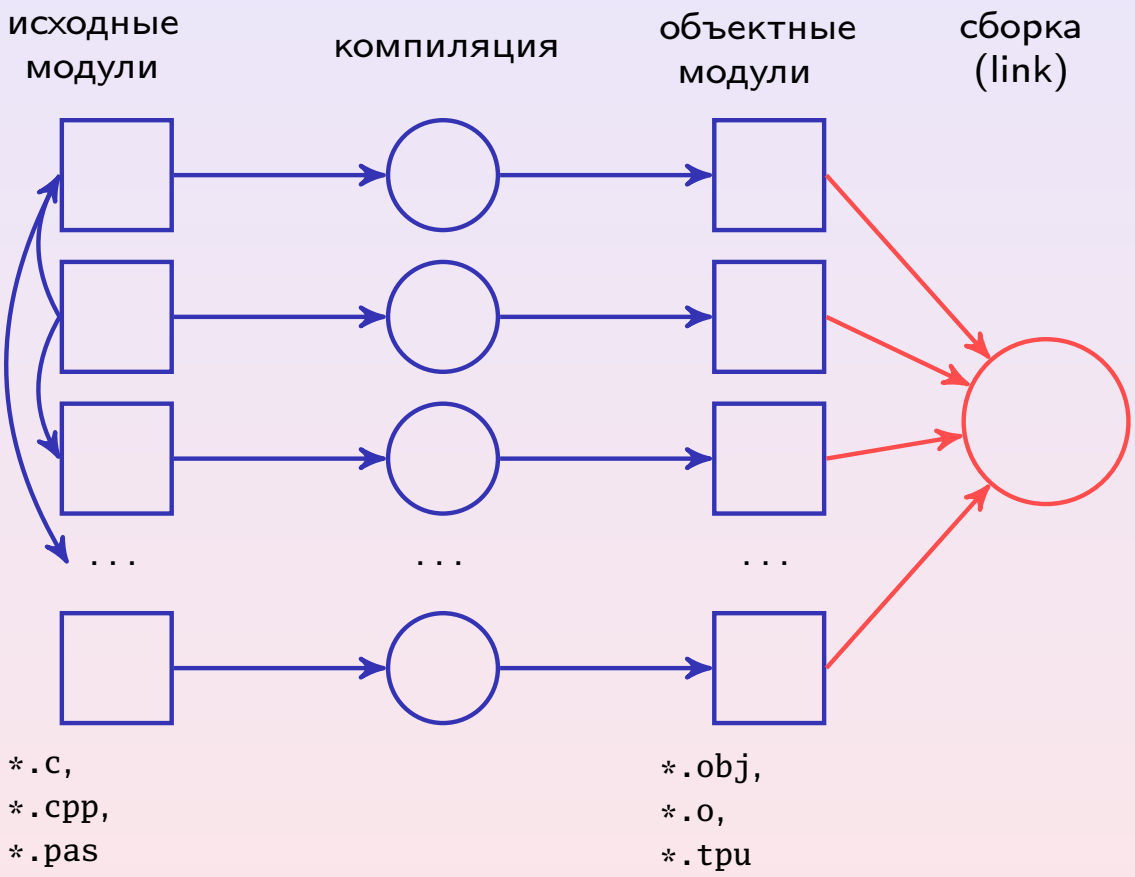

Рис. 1: Схема модульного подхода

Модульный подход к проектированию ПО

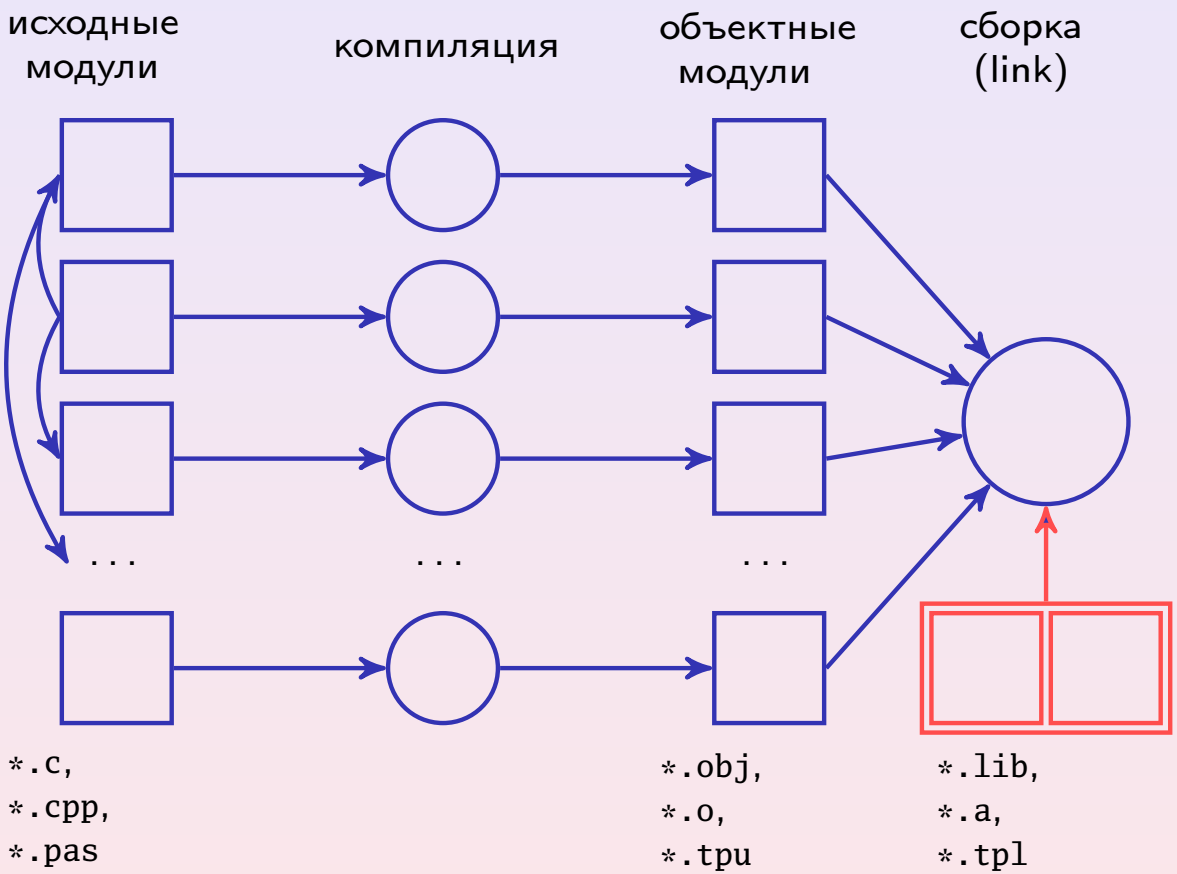

Рис. 1: Схема модульного подхода

Модульный подход к проектированию ПО

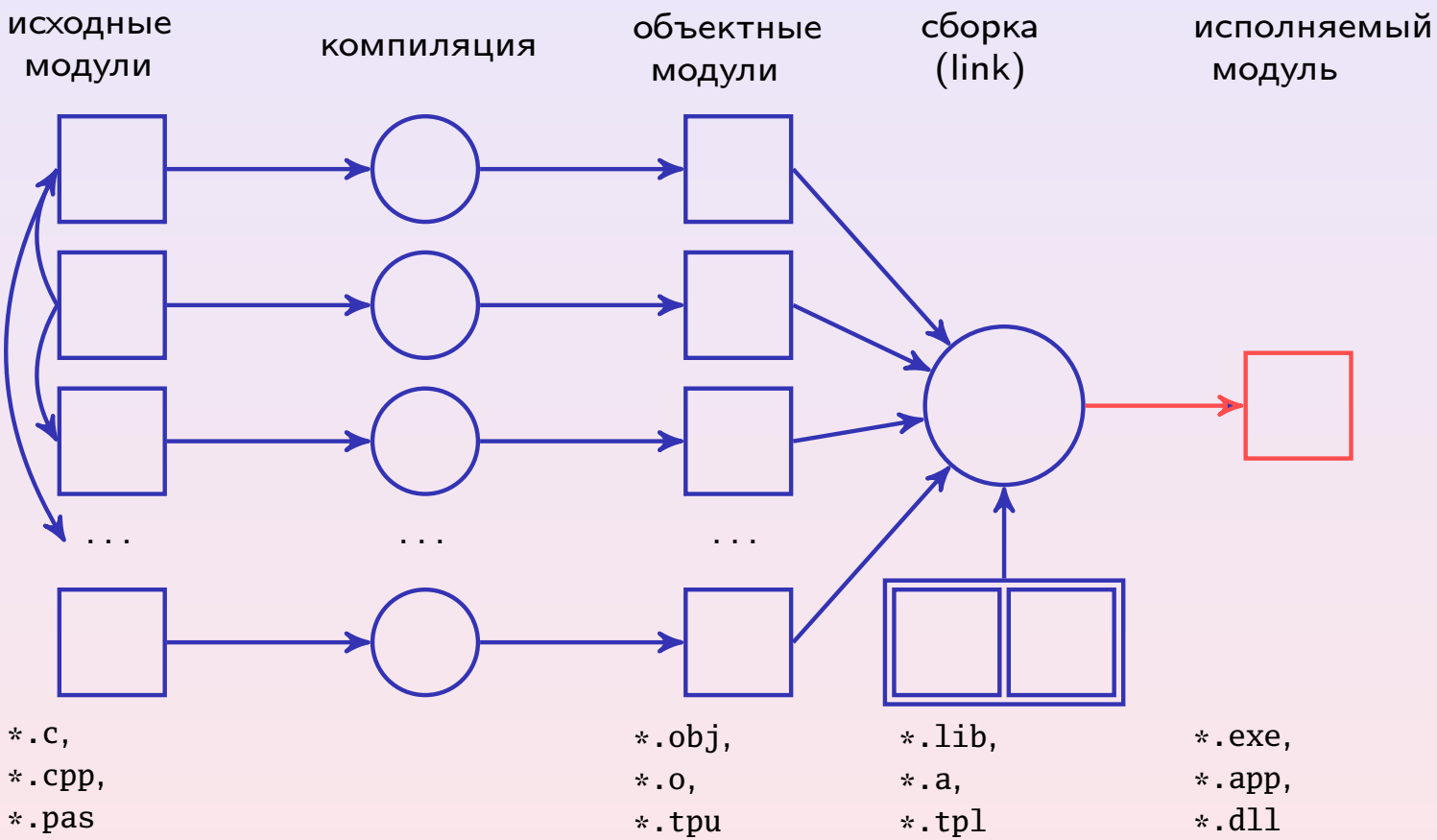

Рис. 1: Схема модульного подхода

Правила для разработчиков

Правила создания транслируемых модулей

- Для ускорения компиляции и сборки проекта, а также для упрощения повторной используемости модулей необходимо уменьшать количество зависимостей между модулями (в т. ч., от стандартных библиотек).
- Необходимо, чтобы каждый модуль был как можно более универсален.
- В каждый модуль необходимо помещать логически взаимосвязанные описания.

Правила для разработчиков

Правила создания транслируемых модулей

- Для ускорения компиляции и сборки проекта, а также для упрощения повторной используемости модулей необходимо уменьшать количество зависимостей между модулями (в т. ч., от стандартных библиотек).
- Необходимо, чтобы каждый модуль был как можно более универсален.
- В каждый модуль необходимо помещать логически взаимосвязанные описания.

Правила для разработчиков

Правила создания транслируемых модулей

- Для ускорения компиляции и сборки проекта, а также для упрощения повторной используемости модулей необходимо уменьшать количество зависимостей между модулями (в т. ч., от стандартных библиотек).
- Необходимо, чтобы каждый модуль был как можно более универсален.
- В каждый модуль необходимо помещать логически взаимосвязанные описания.

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.

Объявления функций

Объявление функции, не являющееся определением

Заголовок функции, после которого идёт символ «;» является объявлением, но не определением.

Пример (объявление)

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

Определения функций

Определение функции

Заголовок функции, после которого идёт её тело в фигурных скобках, является определением.

Примеры (объявление и 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.

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 директивы препроцессора, предотвращающие включение содержимого файла при его повторном подключении:

Правила создания заголовочных файлов

Правила

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

Правила создания заголовочных файлов

Правила

- Помещаются объявления и определения, необходимые в нескольких транслируемых модулях.
- Стандартное расширение (`h`, `hpp`).
- В начале и конце вставляются 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__

/* Здесь идут некоторые объявления и определения */

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

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

Правила создания заголовочных файлов (продолжение)

a.cpp

```
#include "b.h"

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

b.cpp

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

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

Правила создания заголовочных файлов (окончание)

Правила (окончание)

- В одном файле должны группироваться взаимозависимые (возможно, логически) объявления и определения, которые, скорее всего, будут все одновременно нужны (или почти все) в некотором транслируемом модуле. Иначе разные объявления и определения разумнее поместить в разных файлах.

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

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

Пример

```
#define PI 3.1415926
```

Объекты, помещаемые, в заголовочные файлы (C)

Помещаемые объекты

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

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 (заголовки) функций.
- Директивы подключения заголовочных файлов с описаниями, необходимыми для описаний текущего файла.

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 (заголовки) функций.
- Директивы подключения заголовочных файлов с описаниями, необходимыми для описаний текущего файла.

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 (заголовки) функций.
- Директивы подключения заголовочных файлов с описаниями, необходимыми для описаний текущего файла.

Пример

```
#include <stddef.h>

/*
  size_t определён в stddef.h
*/

size_t get_size();
```

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 (**namespace**, могут быть вложены) кроме макроопределений.

Пример

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

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 (**namespace**, могут быть вложены) кроме макроопределений.

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.
- Определения всех остальных шаблонов.
- Пространства имён (**namespace**, могут быть вложены) кроме макроопределений.

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 (**namespace**, могут быть вложены) кроме макроопределений.

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 (**namespace**, могут быть 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;`).

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;`).