Распределение кода по файлам — конспект темы

Объявление vs определение

Объявление функции или класса — ответ на вопрос «Что это такое?», а определение — ответ на вопрос «Как это работает?».

Функция с телом — это и объявление, и определение, но функция без тела — это только объявление, и без определения код не заработает. Компилятор проверяет файл строчку за строчкой.

Это верно не только для функций, но и для классов. Нельзя создавать объекты, использовать методы класса и делать с ним что-либо ещё до того, как компилятор узнает о его существовании и о том, что этот класс умеет.

Директива #include

Код нужно делить на файлы

- чтобы в нём было проще ориентироваться;
- чтобы не перекомпилировать весь проект, а только тот файл, где произошло изменение.

Для включения файла используют кавычки:

```
#include "builder.h"
```

А для включения стандартных библиотек — скобки-уголки:

```
#include <iostream>
```

Заголовочные файлы и файлы с реализацией

Файлы с расширением .h или реже .hpp называются **заголовочными**, а файлы с расширением .cpp — **файлами с реализацией**.

Заголовочные файлы содержат объявления функций, а файлы с реализацией — определения функций.

Заголовочных файлов с одной функцией практически не бывает. Обычно в них гораздо больше информации. Поэтому принято включать заголовочный файл в соответствующий файл с реализацией и называть их одинаково, но с разными расширениями.

Механизм сборки многофайловых проектов

Этап 1. **Препроцессинг**

Компилятор читает код как текст, не вникая в суть. Он проходит файл за файлом и обрабатывает директивы препроцессора. Пример директивы — `include`. Она просто подменяется текстом из другого файла. Основной признак директивы — знак `#`.

На выходе получаются файлы с расширением .ii или просто .i.

Этап 2. **Компиляция**

Файлы переводятся с языка C++ в команды для процессора. На выходе получаются объектные файлы с расширением .o.

Этап 3. **Компоновка**

Объектные файлы связываются в одну программу. Этим занимается компоновщик.

Проблема двойного включения

Директива препроцессора `#pragma once` оберегает от двойного включения одного и того же заголовочного файла. Если файл был явно или неявно включён в файл, второй раз включаться он уже не будет. Нет повторного определения классов — меньше кода.

Добавляйте `#pragma once` во все заголовочные файлы. В IDE для заголовочных файлов можно создать шаблон.

Независимость заголовочных файлов

Иногда для класса нужен #include другого класса. Можно добавить его в заголовочный файл, а можно в реализацию.

Включайте файлы только там, где они действительно используются. Не создавайте избыточных включений.

Каждый заголовочный файл должен включать в себя всё необходимое для своей работы. Тогда не случится ситуация, когда проект меняется в одном месте, а ломается в другом. Чем более инкапсулированы файлы, тем более гибким для изменений станет проект.

Делим проект на файлы

`using namespace std` действует только в своей области видимости. Если объявить его глобально, он начинает действовать везде.

Если вам нужно `using namespace std` в заголовочном файле, сделайте это в небольшом блоке. Так вы ограничите область видимости.

К файлам с реализацией это не относится. Они не включаются в другие файлы, и там смело можно писать `using namespace std`.

Если используете функцию `abs` в заголовочном файле, убедитесь, что вызываете её `std::abs`. Так вы разграничите её с одноимённой функцией `abs` в глобальном пространстве, которая принимает аргументы типа `int`.