Основы языков C/C++

Кафедра ЭО

Петрухин О.М.

Многофайловые проекты

Зачем разделять программу на файлы?

1. **Удобство поддержки**: Когда счёт строк кода идёт на сотни, с программой становится сложно работать
2. **Удобство разработки**: Большие программы легче поддерживать, если их разделить на логические части.
3. **Повторное использование кода**: Можно использовать одни и те же функции или классы в разных проектах.
4. **Сокращение времени компиляции**: Изменения в одном файле не требуют перекомпиляции всего проекта.
5. **Коллаборация**: Разные разработчики могут работать над разными частями программы.

Какие бывают файлы?

В C++ в основном используется два вида файлов для кода:

- **Исходные файлы** (**Source files**)

  Это файлы с расширением **.cpp**. В таких файлах размещается основная часть кода.

  Содержат реализацию: тела функций, методов классов.

- **Файлы заголовков** (**Header files**, ещё известны как **заголовочные файлы**)

  Это файлы с расширением **.h**. Содержат объявления: прототипы функций, классов, структур и **не содержат реализации**, а также содержат **определения** классов **без реализаций** их методов

Пример

Представим, что есть функция, которую мы хотим вынести в новый файл. Например, функция power_2, которая вычисляет квадрат числа

```
int power_2(int x){ // вот эту функцию мы хотим вынести в новый файл
    return x * x;
}

int main(){
    std::cout << power_2(5) << std::endl; // 25
}
```

Пример

Создадим файл “math_sqrt.cpp”, перенесём функцию туда и попробуем собрать

```cpp
// файл math_power.cpp
int power_2(int x){ // вот эту функцию мы хотим вынести в новый файл
    return x * x;
}

// основной файл (например, source.cpp - далее это будет основной файл)
int main(int argc, char** argv){
    std::cout << power_2(5) << std::endl; // 25
}
```

Программа не соберётся

Будет выведена ошибка “идентификатор power_2 не определён”

Пример

В данном случае, код соберётся и отработает правильно.

```
// файл math_power.cpp
int power_2(int x){ // вот эту функцию мы хотим вынести в новый файл
    return x * x;
}

// основной файл (например, source.cpp - далее это будет основной файл)
int power_2(int x); // объявляем функцию
int main(){
    std::cout << power_2(5) << std::endl; // 25
}
```

Однако, так никто не делает, по следующим причинам:

- **Высока вероятность ошибки.**

  Вам придётся вручную писать определения для всех функций, которые вам нужны, со всеми типами аргументов, перегрузками и пр.

- **Очень плохо масштабируется.**

  Часто код пишется в отдельных файлах для того, чтобы можно было его потом легко переиспользовать

Директива #include

Общепринятым подходом для разнесения кода по разным файлам является использование **заголовочных файлов (header files)**. Заголовочные файлы **подключаются** с помощью директивы **#include**.

Директива **#include** в C++ используется для включения содержимого файла в программу во время компиляции. Попросту говоря, на место директивы копируется указанный файл как он есть.

При этом заголовочный файл должен содержать только ту информация, которая необходима для использования функций, классов и переменных, а реализация должна быть вынесена в соответствующий .cpp файл

Директива #include

Существует два вида синтаксиса для директивы **#include**:

```
#include <header.h>
#include "header.h"
```

Разница между этими вариантами заключается в том, где компилятор ищет указанный заголовочный файл.

`#include "header.h"` используется для подключения пользовательских (локальных) заголовочных файлов. Компилятор сначала ищет файл в текущей директории (обычно это директория, где находится исходный файл, содержащий директиву `#include`).

`#include <header.h>` используется для подключения стандартных или сторонних библиотечных заголовочных файлов. Компилятор ищет файл только в стандартных системных путях (указанные в настройках компилятора).

В общем случае при использовании стандартных библиотек (таких как `iostream`) используется синтаксис с угловыми скобками, а при использовании своих локальных заголовочных файлов используется синтаксис с кавычками.

Пример

Вернёмся к нашему примеру. Правильная реализация:

```
// файл math_power.h
int power_2(int x); // объявление функции power_2

// файл math_power.cpp
#include "math_power.h" // в cpp файлах обычно включают свой заголовочный файл
int power_2(int x){ // вот эту функцию мы хотим вынести в новый файл
    return x * x;
}

// основной файл (например, source.cpp - далее это будет основной файл)
#include "math_power.h"

int main(int argc, char** argv){
    std::cout << power_2(5) << std::endl; // 25
}
```

Многократное включение одного заголовочного файла

Проблема. При многократном включении одного заголовочного файла возникают ошибки повторного определения.

При сборке возникнет ошибка: “Переопределение структуры foo”

```
// файл a.h
struct foo{
    int field;
};

// файл b.h
#include "a.h"

// файл source.cpp
#include "a.h"
#include "b.h"
int main()
{ }
```

Так как фактически код файла source.cpp будет выглядеть так:

```
// файл source.cpp
struct foo{
    int field;
};

struct foo{
    int field;
};

int main()
{ }
```

Решение проблемы

Для предотвращения ошибок повторного определения используются специальные механизмы защиты от многократного включения заголовочных файлов. Основные подходы:

1. Include Guards

Это стандартный способ защиты заголовочных файлов. Он заключается в использовании макросов препроцессора для проверки, был ли файл уже включен.

```
// файл a.h
#ifndef A_H
#define A_H

struct foo{
    int field;
};

#endif // A_H
```

`#ifndef A_H` Проверяет, определен ли макрос `A_H`.

Если макрос не определен, то весь код между `#ifndef` и `#endif` включается. Если макрос уже определен (файл был включен ранее), то код пропускается.

Решение проблемы

2. #pragma once

Альтернативный способ защиты заголовочных файлов, поддерживаемый большинством современных компиляторов. Он проще в использовании:

```
// файл a.h
#pragma once

struct foo{
    int field;
};
```

Компилятор гарантирует, что этот файл будет включен только один раз.

Использование директивы является более предпочтительным, чем include guards

КАИ
Крылья для жизни!

Спасибо за внимание!