MULTIMODULE

Минимум о многомодульных программах, компиляции, компоновке и правиле одного определения

К. Владимиров, Intel, 2019
mail-to: konstantin.vladimirov@gmail.com

Переиспользование кода

- Предположим, что вы написали очень интересную программу

```
// --- файл myprog.c ---

// ipow возводит n в степень x
unsigned long long
ipow(unsigned n, unsigned x) {
  // тут очень умная реализация
}

int main () {
  // тут основная программа, использующая ipow
}
```

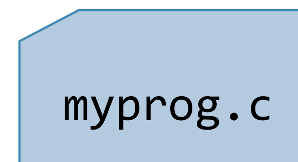
```
> gcc myprog.c –o myprog
```

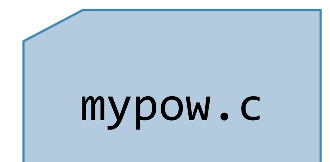
- Она работает, но вы обнаружили, что функция ipow может быть вам нужна и в других программах, т.е. может быть переиспользована

Выносим функцию в модуль

- Вы можете сделать отдельный модуль с функцией ipow

```
// --- файл mypow-1.c ---

// ipow возводит n в степень x
unsigned long long ipow(unsigned n, unsigned x) {
  // тут очень умная реализация
}
```






```
> gcc myprog-1.c mypow-1.c –o myprog
```

- В модуле myprog вам нужно только определение

```
// --- файл myprog-1.c ---

unsigned long long ipow(unsigned n, unsigned x); // declaration

int main () {
  // тут основная программа, использующая ipow
}
```

- Это работает. Все ли видят проблемы с таким подходом?

Обсуждение

- Возникает ощущение, что определение функции никак не связано с её объявлением

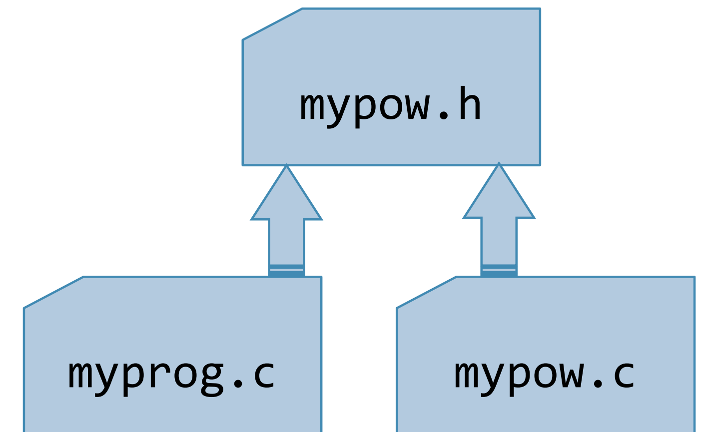
```
// --- файл mypow-1.c ---

// ipow возводит n в степень x
unsigned long long ipow(unsigned n, unsigned x) {
  // тут очень умная реализация
}

// --- файл myprog-2.c ---

unsigned ipow(unsigned n, unsigned x); // oops!
```

- Если они не совпадут, это приведёт к непредсказуемым результатам

Заголовочные файлы

- Выход это составить заголовочный файл с определением и включить его и в файл с реализацией и в файл с использованием

```
// --- файл mypow.h ---

unsigned long long ipow(unsigned n, unsigned x);

// --- файл mypow-2.c ---

#include "mypow.h"

unsigned long long ipow(unsigned n, unsigned x) {
  // тут очень умная реализация
}

// --- файл myprog-3.c ---

#include "mypow.h"

int main () {
  // тут основная программа, использующая ipow
}
```




```
> gcc myprog-3.c mypow-2.c -o myprog
```

Директива #include

- Вы скорее всего ей уже пользовались

```
#include <stdio.h>
#include "mypow.h"
```

- Всё что она делает, это механически включает текст одного файла в другой
- Можно посмотреть результирующий файл после всех инклудов

```
> gcc myprog-3.c -E –o myprog.i
```

- Вид скобок определяет путь поиска файлов: треугольные скобки – файл ищется по системным путям. Используются только для стандартных библиотек

Объектные файлы

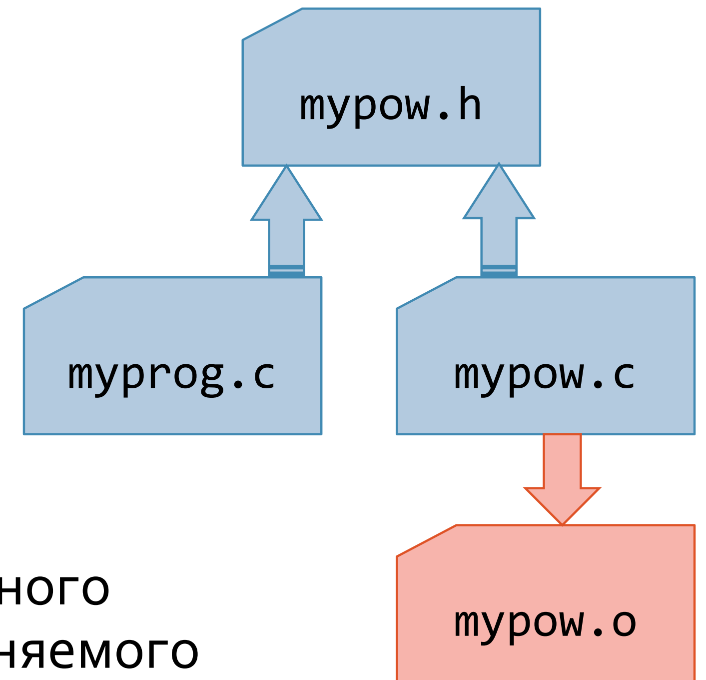
- Вы можете заранее скомпилировать `mypow.c` чтобы не тратить на это время при каждой компиляции использующих его программ

```
> gcc mypow.c -c -o mypow.o
```

- Такой файл называется объектным

```
> gcc myprog.c mypow.o -o myprog
```

- Много объектных файлов можно собрать в библиотеку, но об этом мы поговорим позже
- Говорят, что исходный файл компилируется до объектного и все объектные файлы компонуются вместе до исполняемого



Полная компоновка

- Для симметрии можно прекомпилировать все файлы

```
> gcc mypow.c -c -o mypow.o
> gcc myprog.c -c -o myprog.o
```

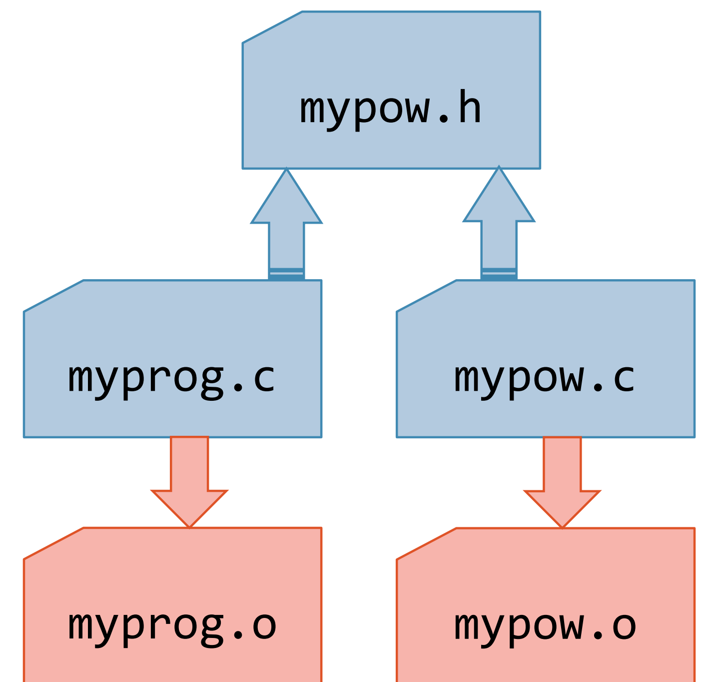
- Теперь скомпоновать (слинковать) исполняемый

```
> gcc myprog.o mypow.o -o myprog
```

- Эти три команды эквивалентны команде

```
> gcc myprog.c mypow.c -o myprog
```

- Но дают больше контроля над процессом



Стражи включения

- Один заголовочный файл может быть включён в тысячи файлов в одном проекте
- Чтобы избежать лишних включений, можно использовать прагму

```
// --- файл mypow.h ---
#pragma once
unsigned ipow(unsigned n, unsigned x);
```

- Позднее мы поговорим о стражах включения подробнее

Экспорт функций

- Функция из одного модуля, которая видна в другом называется экспортируемой (extern)

```
// --- файл mypow.h ---
#pragma once
extern unsigned ipow(unsigned n, unsigned x);
```

- Все функции по умолчанию extern, это слово можно не писать. Если какая-то функция не экспортируется наружу, она помечается static

```
// --- файл mypow.c ---
#include "mypow.h"

static unsigned isqr(unsigned n) { return n*n; }

extern unsigned ipow(unsigned n, unsigned x) {
  // тут очень умная реализация, использующая isqr
}
```

- Не путайте static функции со static переменными внутри функций! Это необъяснимая омонимия

Экспорт переменных

- Экспортировать также можно переменные. Тогда указывать `extern` обязательно

```
// --- файл mypow.h ---
#pragma once

extern double expsq; // e^2
extern unsigned ipow(unsigned n, unsigned x);
```

- Если вы не напишете extern, будет считаться, что вы определили переменную

```
// --- файл mypow.c ---
#include "mypow.h"

/* не extern! */ double expsq = 2.718281828 * 2.718281828;
```

- Избегайте определений в заголовочных файлах

Объявления и определения

- Объявление сообщает имя сущности компилятору

```
extern int a;     // объявление переменной
int foo(int);     // объявление функции
struct S;         // объявление структуры
```

- Определение сообщает подробности (адрес, состав, исходный код) компоновщику

```
int a;                              // определение переменной
int foo(int x) { return x * 2; }    // определение функции
struct S { int x; int y; };         // определение структуры
```

- В языке действует ODR – правило одного определения

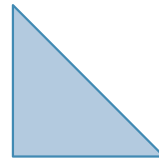
Правило одного определения (ODR)

- ODR гласит, что в программе может быть сколько угодно объявлений, но ровно одно определение для каждой сущности с внешним связыванием (думайте об этом как об extern сущностях)
- Таким образом две разных но одинаково называющихся static функции в двух разных модулях это нормально, а extern – нарушение
- Это нарушение никто не диагностирует и оно может привести к сложным неявным ошибкам
- Поэтому если можно сделать функцию static надо это делать
- Функция main не может быть static и поэтому она всегда одна на все единицы трансляции

Problem TS: площадь треугольников

- Задан файл, в котором сначала указано количество треугольников, а потом перечислены координаты вершин по шесть вещественных чисел (x, y)

```
1
0.0 0.0 0.0 1.0 1.0 0.0
```



- Необходимо написать две программы: первую, которая, считывая этот файл, определяет треугольник максимальной площади и вторую, которая определяет суммарную площадь всех треугольников
- Какие функции вы вынесете в отдельный модуль, чтобы использовать в обоих программах?