# ЛЕКЦИЯ 13.1
# MULTIMODULE

АЛГОРИТМИЗАЦИЯ И
ПРОГРАММИРОВАНИЕ

ЛЕКТОР ФУРМАВНИН С.А.

# ПЕРЕИСПОЛЬЗОВАНИЕ КОДА

- Предположим, что вы написали очень интересную программу

```
// --- файл myprog.cpp ---
// 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
}
```


```
> g++ myprog.cpp -o myprog.exe
```

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


- В модуле myprog вам нужно только определение

```
// --- файл myprog-1.cpp ---
unsigned long long ipow(unsigned n, unsigned x); // declaration
int main () {
  // тут основная программа, использующая ipow
}
```

```
> g++ myprog-1.cpp mypow-1.cpp -o myprog.exe
```

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

# ОБСУЖДЕНИЕ

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

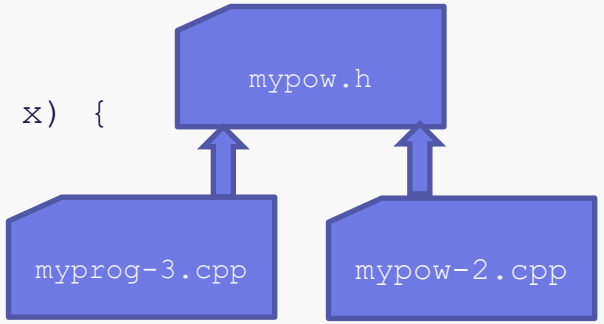
```
// --- файл mypow-1.cpp ---
// ipow возводит n в степень x
unsigned long long ipow(unsigned n, unsigned x) {
  // тут очень умная реализация
}

// --- файл myprog-2.cpp ---
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.cpp ---
#include "mypow.h"
unsigned long long ipow(unsigned n, unsigned x) {
// тут очень умная реализация
}
// --- файл myprog-3.cpp ---
#include "mypow.h"
int main () {
  // тут основная программа, использующая ipow
}
```




```
> g++ myprog-3.cpp mypow-2.cpp -o myprog.exe
```

# ДИРЕКТИВА #INCLUDE

- Вы скорее всего ей уже пользовались

```
#include <iostream>
#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.cpp -E -o myprog.i
```

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

# ОБЪЕКТНЫЕ ФАЙЛЫ

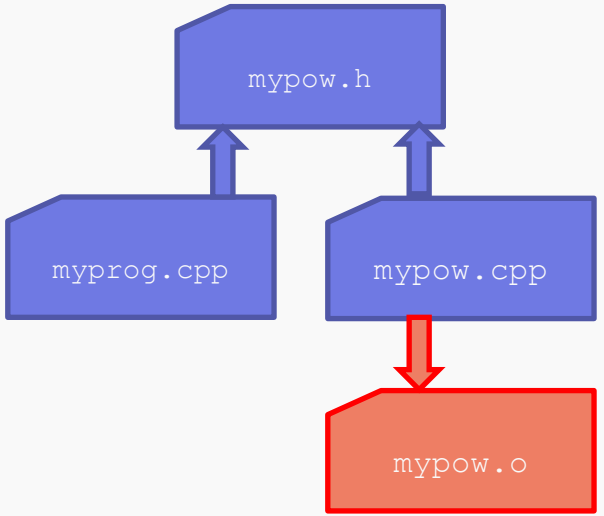
- Вы можете заранее скомпилировать mypow.c чтобы не тратить на это время при каждой компиляции использующих его программ

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

- Такой файл называется **объектным**

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

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

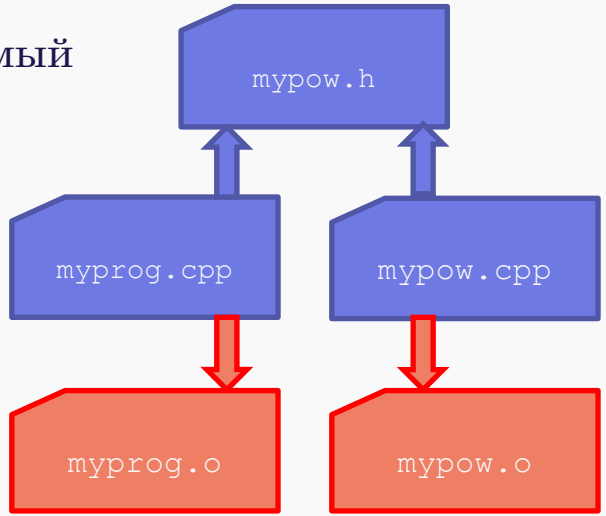
- Теперь скомпоновать (слинковать) исполняемый

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

- 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.cpp ---
#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

# ЗАДАЧА. 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
  - вторую, которая определяет суммарную площадь всех треугольников

- Какие функции вы вынесете в отдельный модуль, чтобы использовать в обеих программах

# РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА


1. Дорохова Т.Ю., Основы алгоритмизации и программирования : учебное пособие для СПО / Т.Ю. Дорохова, И.Е. Ильина. – Саратов, Москва : Профобразование, Ай, Пи Ар Медиа, 2022. – 139 с.
2. Кудинов Ю.И., Основы алгоритмизации и программирования : учебное пособие для СПО / Ю.И. Кудинов, А.Ю. Келина. – 2-е изд. – Липецк, Саратов: Липецкий государственный технический университет, Профообразование, 2020. – 71 с.
3. Дональд Кнут, Искусство программирования. Том 1. Основные алгоритмы / Ю.В. Козаченко. - 3-е изд – Москва, Санкт-Петербург: ВИЛЬЯМС, 2018. – 721 с.