Ключевое слово static. Это слово многоцелевое:

1.Локальные static переменные. Например локальная переменная некоторой функции. void f () { int c=0; c++; return c; } – неудачная попытка посчитать, сколько раз вызвана функция f. Если объявить переменную как static, то она создастся и проинициализируется один раз. int f() { static int c=0; c++; return c; } – на втором вызове вернётся уже 2.

Второй пример : int f () { static int c=3; c++; return c;} – при вызове f() вернётся 4, при втором вызове f() вернётся уже 5, т.к. статичная переменная инициализируется один раз. На самом деле локальная static переменная хранится в глобальной области памяти Получается эта локальная статическая переменная в некотором смысле глобальная ☺

2.Глобальные static переменные. Один из способов решения проблемы глобальных переменных в рамках всего проекта – локализация глобальных переменных. 1.c : static int count=0; - переменная count будет глобальной на уровне одного файла 1.c. Теперь в 2.c : можно писать int count=0; - это будет глобальная переменная для всего проекта, она не будет пересекаться с count в 1.c.

Технически на этапе линковки происходит два вида линковки: внутренняя (internal) и внешняя (external). Среда разработки и компилятор выдают ошибки с соответствующими названиями (ошибки линковки). Таким образом, когда линковщик получает объектные модули, он сначала в каждом модуле отдельно проводит линковку. То есть он создаёт таблицу для конкретного модуля, происходит разрешение имён на уровне одного файла исходного кода.

Пример: func.h : static int count=0; 1.c : #include “func.h” 2.c : #include “func.h тогда в каждом файле исходного кода будет своя копия переменной. Зачем нужно такое разделение глобальных переменных? Таким образом мы контролируем то место, где можно изменять эту переменную. Только в этом файле.

3.static-функции. Вообще когда вы объявляете функцию, она автоматически объявляется с ключевым словом extern. То есть все функции по умолчанию имеют внешнюю линковку. void f(){} == extern void f(){}. Если мы хотим запретить использование функции в других модулях, сделать её доступной только в пределах одного файла исходного кода, то делаем её static и получаем внутреннюю линковку : static void f () {}. Зачем это нужно? Допустим мы реализуем библиотеку, которая что-то сложное считает, и мы хотим, чтобы пользователи видели только несколько полезных основных функций. Другие вспомогательные наши функции пусть будут внутренними. Сразу два соображения: во-первых, эти функции не нужны пользователям снаружи, во-вторых, использование этих функций снаружи может всё сломать. Ещё соображение – функция test() может появиться в проекте в десятках файлов разных отделов разработки. То есть проблема во внешней линковке как раз, а делая функцию static мы используем внутреннюю линковку.

Пример: a.h : static int c=0; static void f() {}. Если бы мы её определили без static, то из-за внешней линковки и нескольких .c файлов, включающих этот заголовочный, было бы multiply definition – многократное определение функции f(). С ключевым словом static получаются копии функций f() локальных для каждого файла исходного кода.

В отсутствии классов и ООП наличие таких механизмов разделения областей видимости вполне естественно.