*PE Linker

COFF - формат

- Common Object File Format - стандартный формат объектного файла
- Некоторые поля файла имеют восьмеричный формат
- COFF-формат был сам по себе неплохой отправной точкой, но нуждался в расширении, чтобы удовлетворить потребностям новых операционных систем, таких как Windows NT или Windows 98. Результатом такого усовершенствования явился PE-формат

* Portable Executable - переносимый исполняемый

* Это формат исполняемых файлов, объектного кода и динамических библиотек, используемый в 32- и 64-битных версиях операционной системы Microsoft Windows.
* Формат PE представляет собой структуру данных, содержащую всю информацию, необходимую PE загрузчику для проецирования файла в память.
* PE-файл состоит из заголовка и некоторого набора секций, количество и размер которых зависит от информации, содержащейся в заголовке.

* COFF и PE. В чем различие?

* Компоновщик не превращает объектный файл в исполняемый, а создаёт загрузочный модуль на основе информации, содержащейся в одном или нескольких объектных модулях.

* Другими словами, объектный и исполняемый файлы - это два совершенно разных файла, хотя и содержащие значительный объем одинаковой информации.

Форматы исполняемых файлов							
Windows, DOS, OS/2	.COM	.EXE(MZ/NE/LE/LX/PE)					
Unix	a.out	COFF	ECOFF	ELF	Mach-O	SOM	XCOFF
Прочие	Intel HEX	PEF	SREC				

Шаг 1 - Написание программы

```
.386
.model flat
    extrn MessageBoxA: dword
    extrn ExitProcess: dword
    extrn GetComputerNameA: dword
.code
_start:
    push offset nSize
    push offset lpBuffer
    call GetComputerNameA
    push 40h
    push offset msg
    push offset lpBuffer
    push 00h
    call MessageBoxA
    push 00h
    call ExitProcess
.data
    lpBuffer db 20 dup(0),0
    nSize db 3 dup(0),14h
    msg db 'PE Linker',0
end _start
```

* обязательно должна использоваться модель памяти FLAT (плоская бессегментная модель).
* все внешние функции (в данном случае - функции API) необходимо объявлять с помощью директивы:
  extrn <имя функции>: dword
* имена функций чувствительны к регистру символов!!!
* а…рать из задания!

Адрес загрузки
00400000h

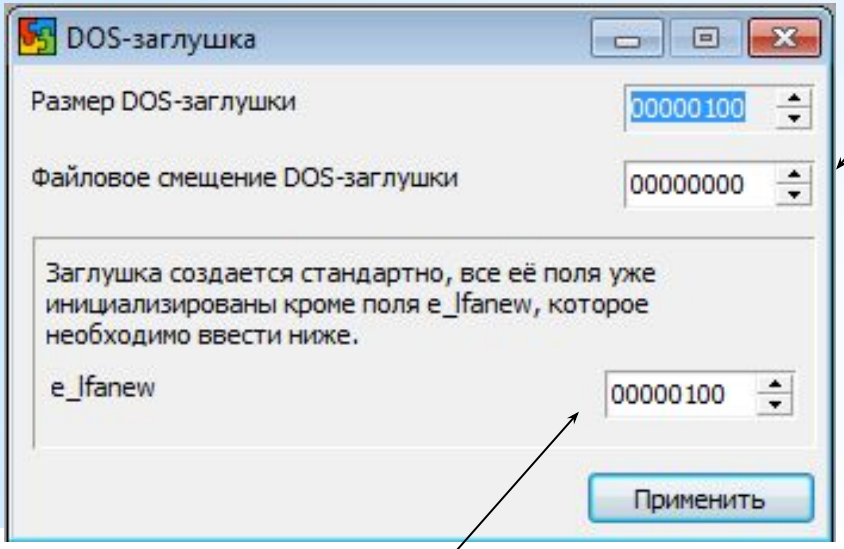
* После написания программы её необходимо откомпилировать с помощью команды:

  ml/coff /c <имя файла>

* Шаг 2 - Создание заголовка PE-файла

* Как и в других исполняемых форматах от Microsoft, заголовок не находится в самом начале файла. Вместо этого несколько сотен первых байтов типичного PE-файла заняты под *заглушку DOS*.
* Эта заглушка представляет собой минимальную DOS-программу, которая выводит что-либо вроде: "Эта программа не может быть запущена под DOS".
* Все это предусматривает случай, когда пользователь запускает программу Win32 в среде, которая не поддерживает Win32, получая при этом приведенное выше сообщение об ошибке.

* Шаг 2 - Создание заголовка PE-файла

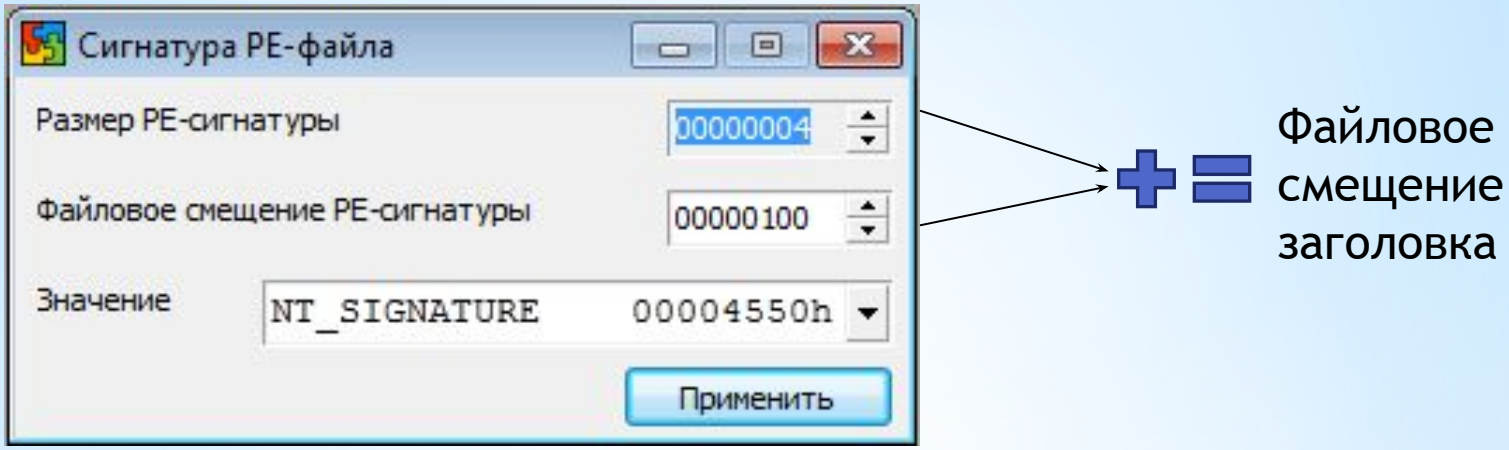


* первый байт отображения файла соответствует первому байту заглушки DOS.

* настоящий заголовок можно обнаружить, найдя его стартовое сме щение, которое хранится в заголовке DOS.

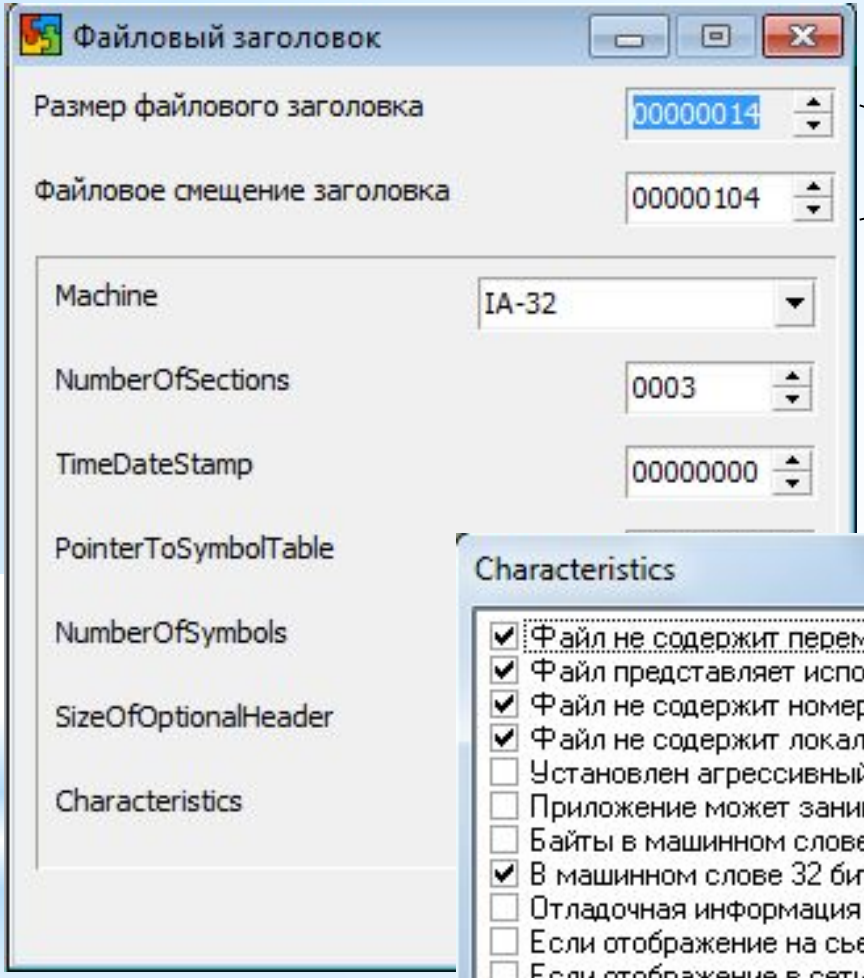
* Поле **e_lfanew** собственно и содержит относительное смещение (RVA) настоящего заголовка PE-файла.

* Шаг 2 - Создание заголовка PE-файла



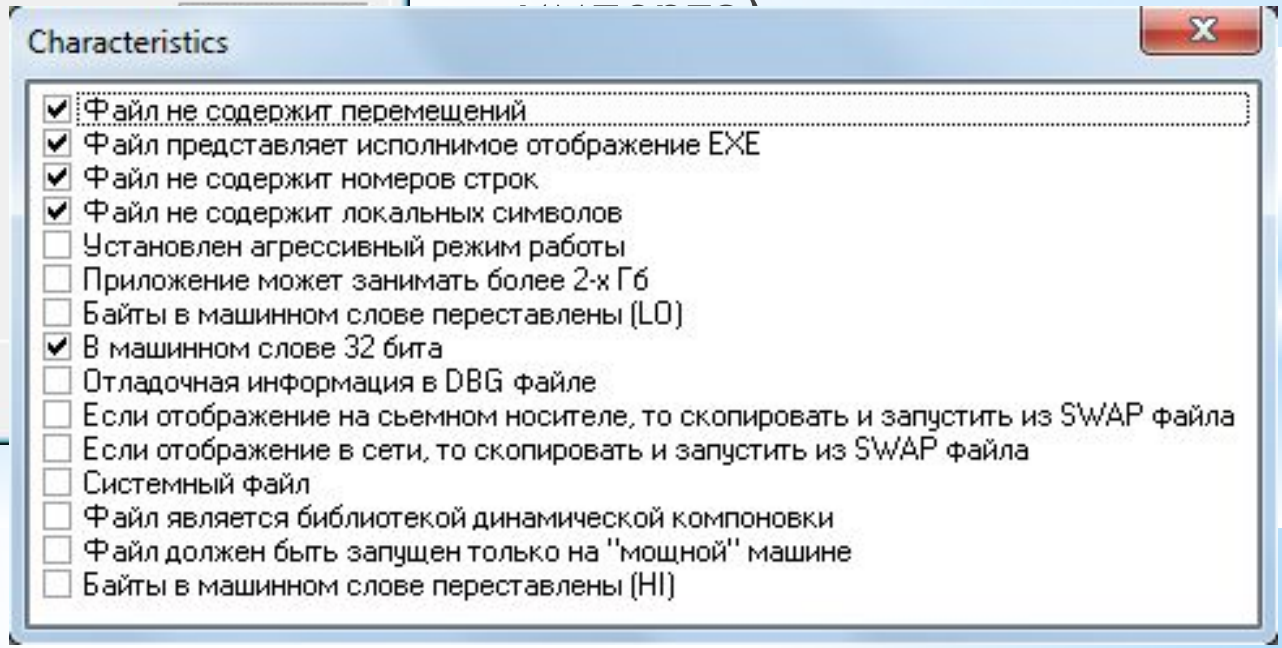
*Поле Signature (сигнатура - подпись), представленное как ASCII код, - это PE00 (два нулевых байта после PE).

* Шаг 2 - Создание заголовка PE-файла

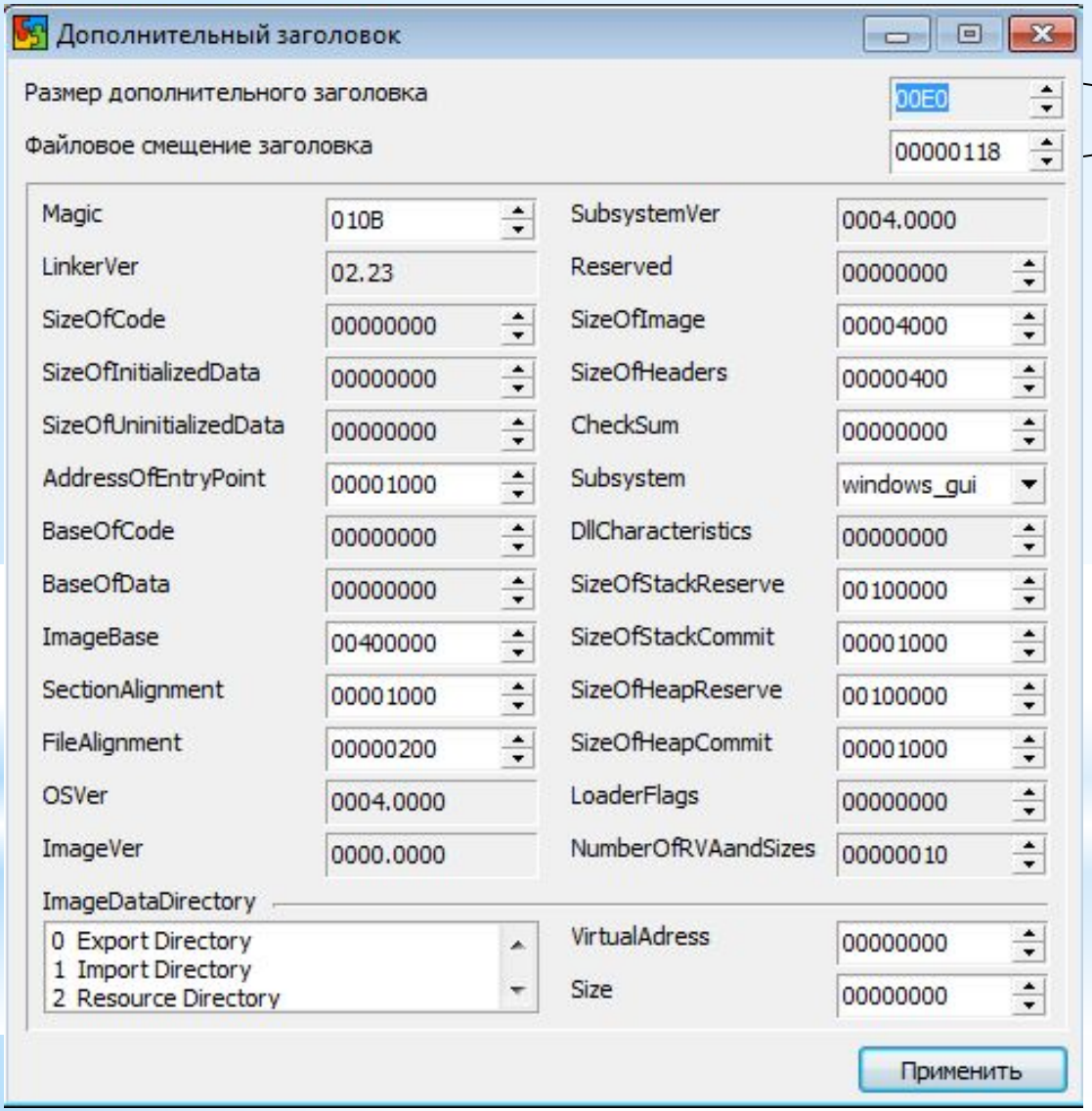


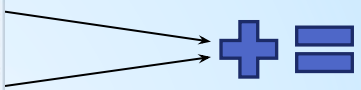
Файловое смещение дополнительного заголовка

*NumberOfSections – кол-во секций = 3 (кода,данных,



* Шаг 2 - Создание заголовка PE-файла





Файловое смещение таблицы секций

* ImageBase – адрес загрузки (см.шаг 1)
* Magic - слово-сигнатура, определяющее состояние отображенного файла(010b-исполняемое отобра-жение). Для 64 разрядной системы равно 020b.

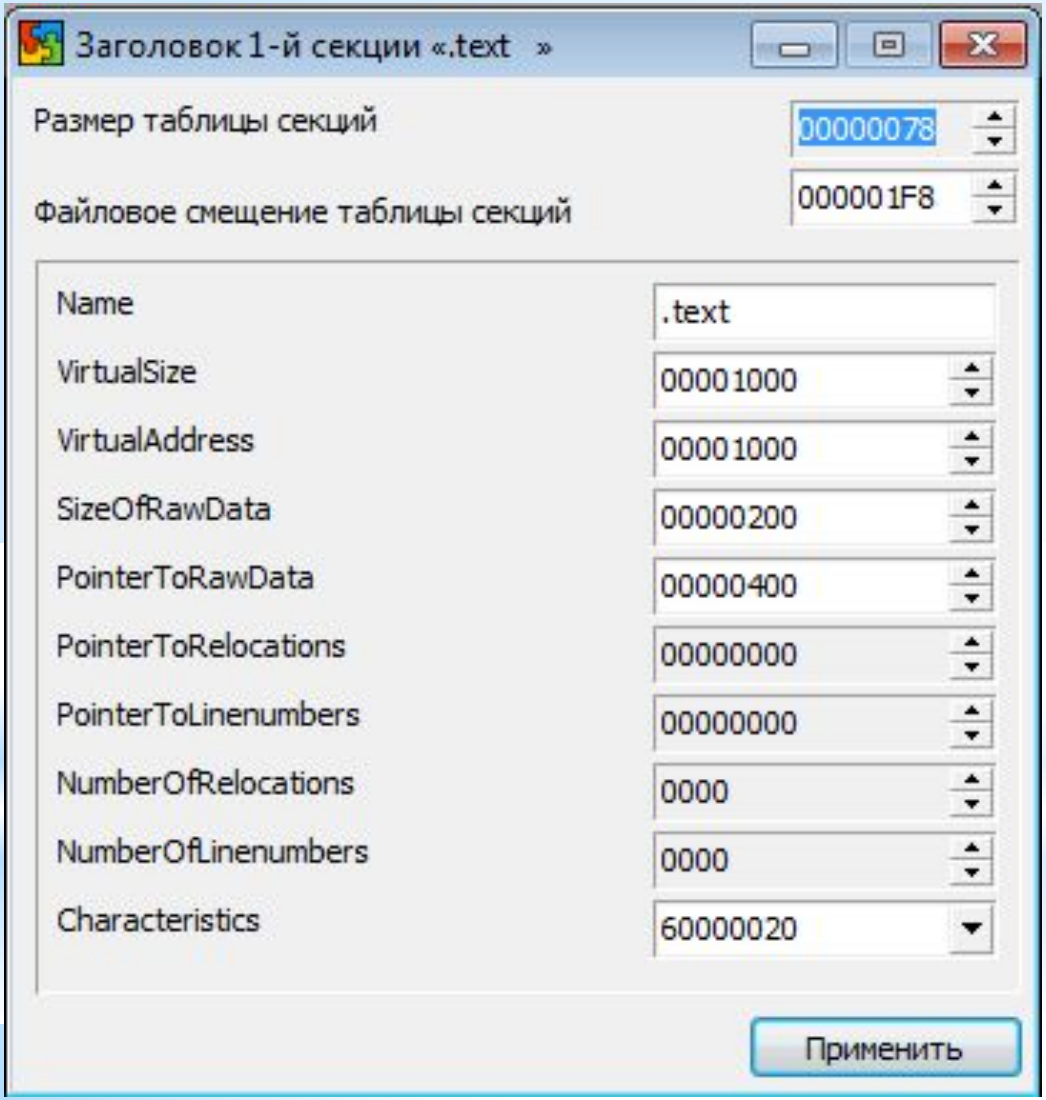
* Шаг 2 - Создание заголовка PE-файла

* AddressOfEntryPoint = 1000 (входная точка главного потока = RVA данных секции кода(.text)
* SectionAlignment ≥ 1000 (const Кратность выравнивания секций в памяти = размер страницы)
* FileAlignment ≥ 200 (const Кратность выравнивания секций на диске = размер сектора винчестера)
* SizeOfImage = VirtualAddress(последней скции) + VirtualSize(последней секции) = 3000+1000=4000
* SizeOfHeaders = 400 (const = размер всех заголовков и таблицы секций)

* Шаг 2 - Создание заголовка PE-файла

* SizeOfStackReserve = 100000 (const = зарезервированный в вирт. пространстве объём для стека главного потока)
* SizeOfStackCommit = 1000 (const = зарезервированный в пространстве физ. памяти объём для стека главного потока)
* SizeOfHeapReserve = 100000 (const = зарезервированный объём для главного хипа)
* SizeOfHeapCommit = 1000 (const = зарезервированный в пространстве физ. памяти объём для главного хипа)

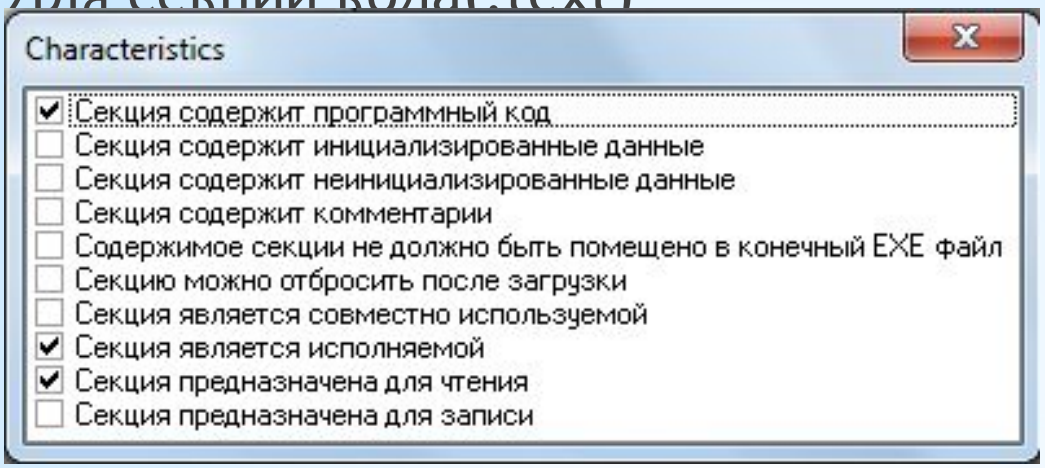
* Шаг 3 - Создание секций PE-файла



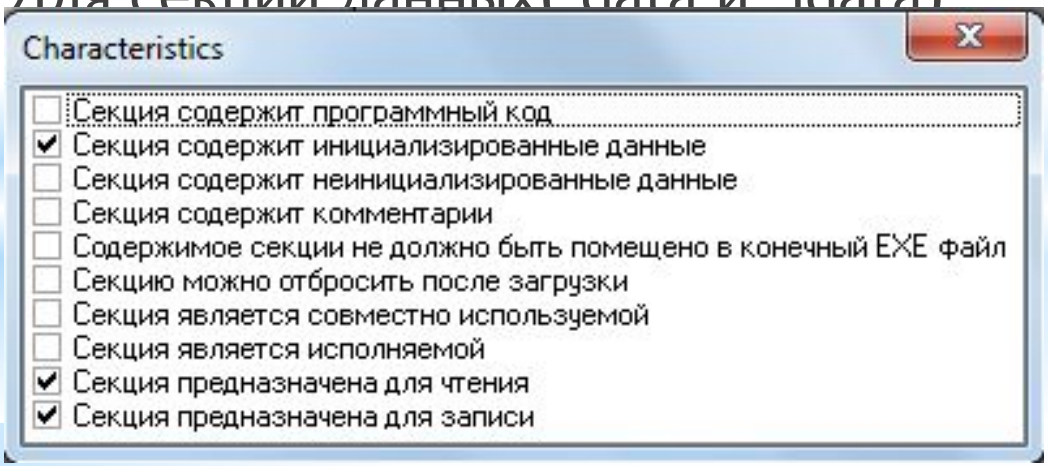
* Name – название секции
* VirtualSize = 1000 (вирт. размер секции)
* VirtualAddress = 1000 + VirtualSize * (номер секции -1) (адрес начала секции в памяти)
* SizeOfRawData = 200 (физ. размер секции)
* PointerToRawData = 400 + SizeOfRowData * номер секции (смещение относительно начала файла)

* Шаг 3 - Создание секций PE-файла

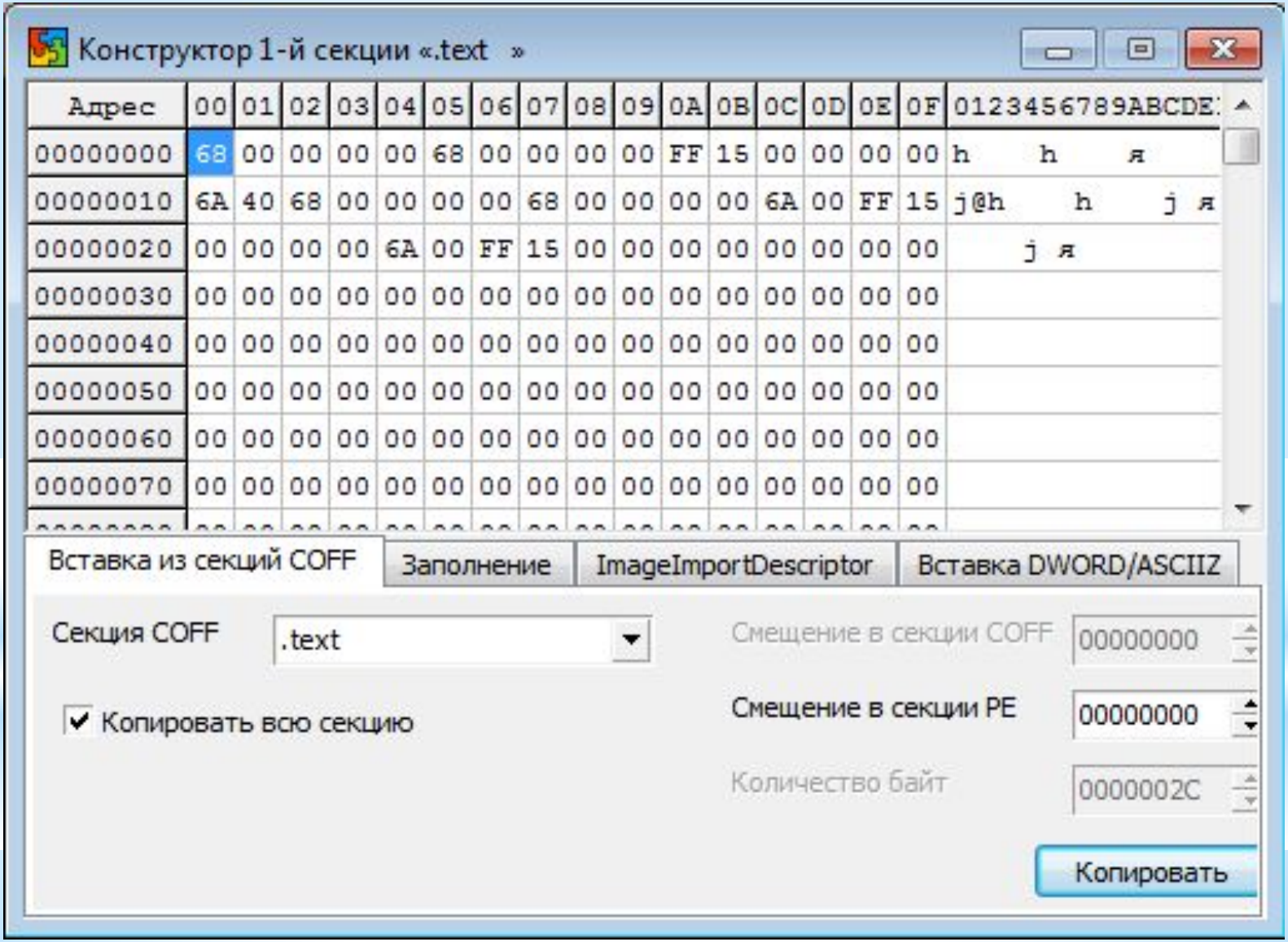
* Для секции кода(.text)



* Для секции данных(.data и .idata)



* Шаг 3 - Создание секций PE-файла



* Шаг 3 - Создание секций PE-файла

*Если это не секция “.idata” то

1) Клик мышкой на ячейку (0;0)
2) В нижней части всплывшего окна выбираем вкладку «Вставка из секции COFF»
3) устанавливаем в поле «Секция COFF» открывшейся панели имя совпадающее с именем этой секции.
4) устанавливаем в поле «Копировать всю секцию» открывшейся панели галочку.
5) Нажимаем кнопку «Копировать».

* Шаг 3 - Создание секций PE-файла

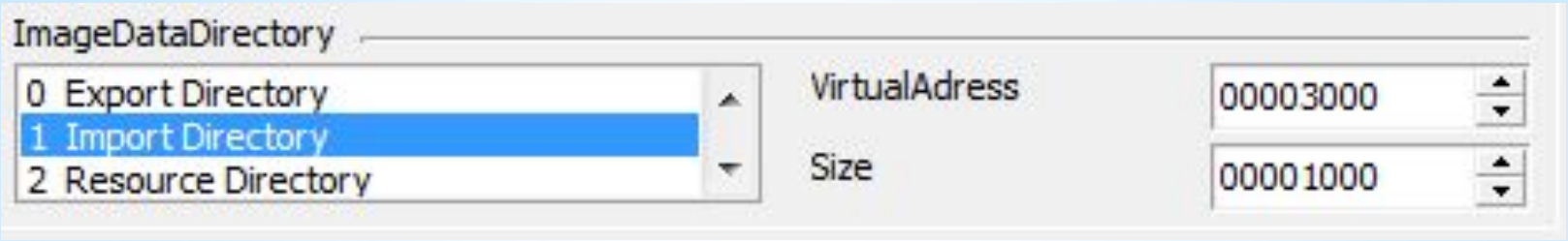
*Что храниться в секции “.idata” ?

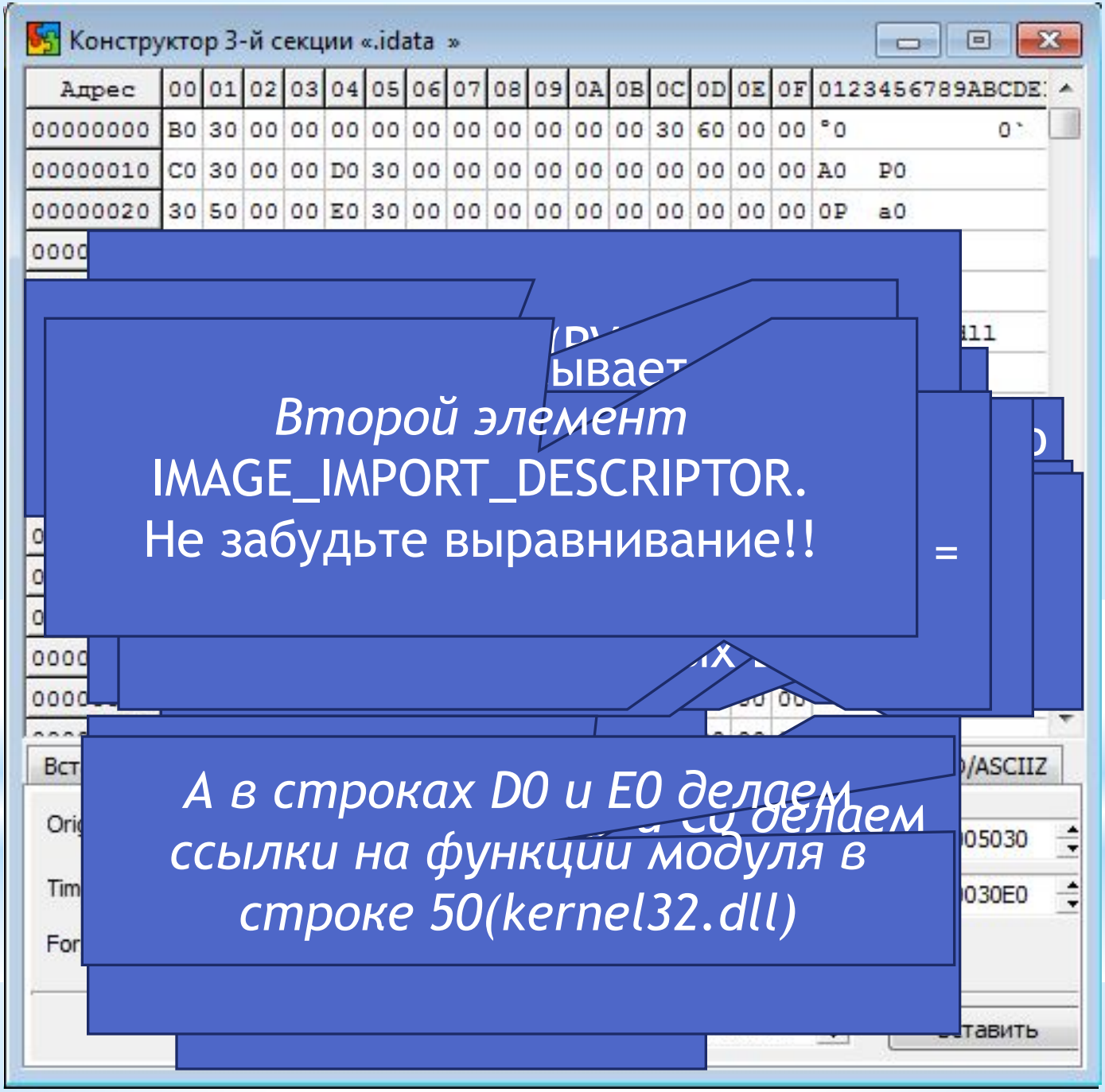
- Перед загрузкой в память информация, хранящаяся в секции .idata PE-файла, содержит информацию, необходимую для того, чтобы загрузчик мог определить адреса целевых функций и пристыковать их к отображению исполняемого файла.
- После загрузки секция .idata содержит указатели функций, импортируемых EXE-файлом или DLL.

* Шаг 3 - Создание секций PE-файла

*Если это секция “.idata” то

*Предварительно подключаем таблицу импорта в заголоке.




Конструктор 3-й секции «.idata »
Адрес 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDE
00000000 B0 30 00 00 00 00 00 00 00 00 00 00 30 60 00 00
00000010 C0 30 00 00 D0 30 00 00 00 00 00 00 00 00 00 00
00000020 30 50 00 00 E0 30 00 00 00 00 00 00 00 00 00 00
Второй элемент
IMAGE_IMPORT_DESCRIPTOR.
Не забудьте выравнивание!!
А в строках D0 и E0 делаем
ссылки на функции модуля в
строке 50(kernel32.dll)

* Шаг 4 - Разрешение статических и внешних ссылок

* Очень важный этап компоновки - разрешение статических и внешних ссылок.
* На этапе компиляции неизвестны реальные адреса переменных и функций API, поэтому компилятор превращает адреса переменных в статические, а адреса функций API - во внешние ссылки.
* Информация о неразрешенных ссылках хранится в двух местах в объектном модуле: в COFF-таблице символов и в списках привязок для каждой секции.

* Шаг 4 - Разрешение статических и внешних ссылок

Для разрешения ссылок для каждой секции COFF-файла используется следующий алгоритм:

1) найти первую, еще не разрешенную ссылку в списке привязок данной секции. Если таких нет, то алгоритм завершен;
2) найти символ в COFF-таблице, на который ссылается данная привязка;
3) если символ является внешним(тип EXTERNAL), то перейти к пункту 9;
4) если данный символ имеет тип STATIC, то данная ссылка является разрешимой;
5) найти секцию PE, соответствующую секции с номером SectionNumber COFF-Файла;
6) сосчитать неизвестный адрес по следующей формуле:

Искомый_адрес = Адрес_загрузки(см.шаг 1) +
RVA_секции_из_пункта_5_алгоритма +
Поле_Value_из_COFF-символа

* Шаг 4 - Разрешение статических и внешних ссылок

7) в секции PE-файла, соответствующей данной секции COFF-файла, по смещению Address из привязки вставить значение, полученное в пункте 6 алгоритма;

8) перейти к пункту 1.

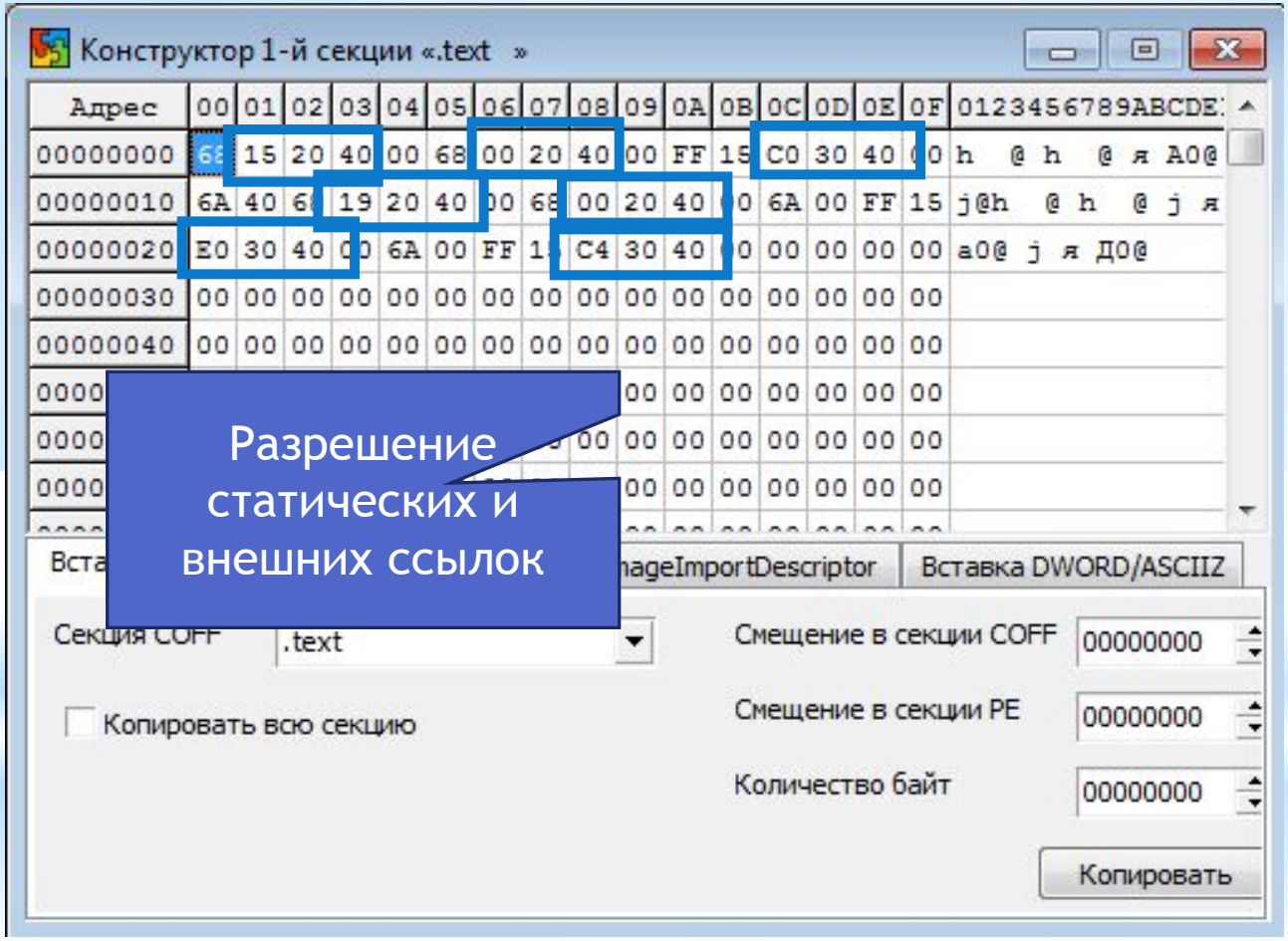
9) сосчитать неизвестный адрес по следующей формуле:

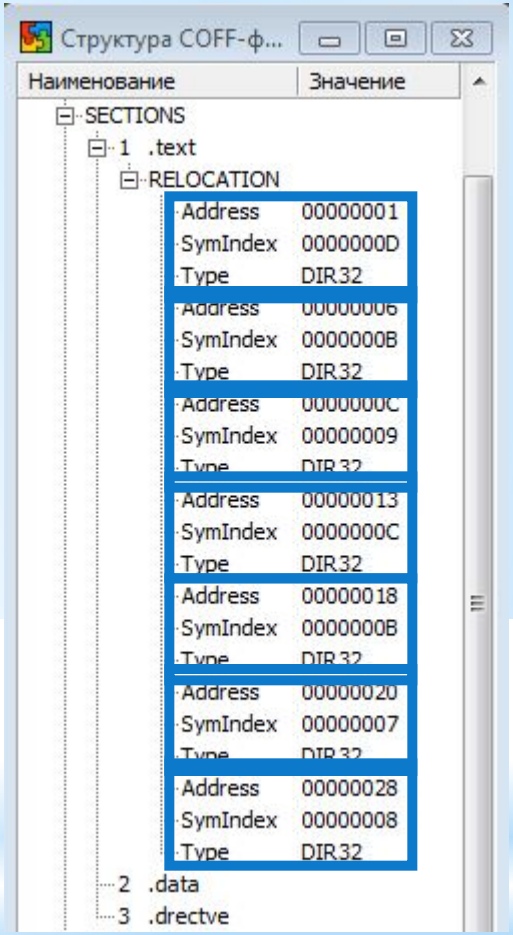
Искомый_адрес = Адрес_загрузки + RVA элемента массива FirstThunk описывающего данную функцию

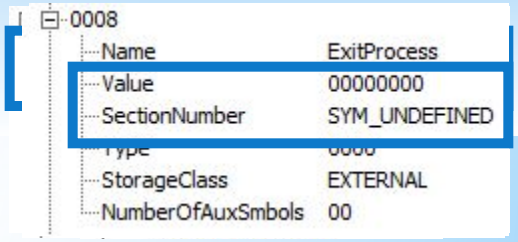
10) в секции PE-файла, соответствующей данной секции COFF-файла, по смещению Address из привязки вставить значение, полученное в пункте 5 алгоритма;

11) перейти к пункту 1.

* Шаг 4 - Разрешение статических и внешних ссылок







* Шаг 5 – Компановка

* Если все шаги сделаны правильно, то после компоновки (CTRL+F9) в каталоге проекта появится исполняемый файл, работоспособность которого необходимо проверить, запустив его на выполнение (F9).
* Запустить программу в дебаггере.

* УДАЧНОЙ ЛАБОРАТОРНОЙ!!!)