сишные трюки
(11h выпуск)

крис касперски ака мыщъх, a.k.a. souriz, a.k.a. nezumi, a.k.a. elraton, no-email

сегодня мы поговорим о статических/динамических массивах нулевой длинны. "а разве бывают такие?" — спросит недоверчивый читатель. не только бывают, но и утверждены стандартом, а так же активно используются всеми, кто о них знает (правда, знают о них немногие, и разработчики компиляторов в том числе).

трюк первый — статические массивы на стеке

Зачем может понадобится создавать статический массив нулевой длинны? Выражение типа "`char c[0]`" не имеет смысла! Однако… в некоторых ситуациях оно бывает _очень_ полезно. Допустим, мы имеем определение DATA_LEN с допустимыми значениями от 0 (no data) до… XXL. Тогда конструкция "`char c[DATA_LEN]`" при DATA_LEN == 0 приведет к ошибке компиляции, даже если мы не собираемся обращаться к массиву "c" по ходу исполнения программы. А усложнять алгоритм, добавляя лишние ветвления и загромождая листинг командами препроцессора — не хочется.

Вся хитрость в том, что если обернуть статический массив структурой, то компилятор, проглотит ее не задумываясь! Как раз то, что нам нужно!!! Рассмотрим следующий код:

```
#define DATA_LEN       0       // нет данных

struct ZERO                    // структура со статическим массивом нулевой длинны
{
        char c[DATA_LEN];      // массив нулевой длины
};

main()
{
        // объявляем структуру с массивом нулевой длины
        struct ZERO zero;

        // печатаем размер структуры ZERO и ее экземпляра zero
        printf("%x %x\n", sizeof(struct ZERO), sizeof(zero));

        // присваиваем значение первой ячейке массива нулевой длины!!!
        *zero.c = 0x69;

        // выводим это значение на экран
        printf("0%Xh\n", *zero.c);
}
```

Листинг 1 статический массив нулевой длинны на стеке

Этот код компилируется всеми компиляторами без исключения, причем, работает _правильно_ (хотя и не обязан это делать). В частности, при компиляции Си-программы, Microsoft Visual C++ утверждает, что размер структуры ZERO равен 4 байтам, но если изменить расширение файла с ".c" на ".cpp", мы получим… 1 байт.

GCC во всех случаях дает нам 0 байт, что логично, но неправильно, поскольку _все_ 32-битные компиляторы _реально_ резервируют как минимум 4 байта под локальные переменные любых видов, т. к. это необходимо для выравнивания стека (и дизассемблерные листинги наглядно подтверждают это!).

Следовательно, мы можем не только создавать статические массивы нулевой длины, но еще и (пускай не без предосторожностей) использовать их. Например, в качестве вступительных тестов для новичков. Шутка! Но своя доля истины в ней есть. Обычно программисты, не желающие, чтобы их отстранили от проекта, добавляют в исходный код немного "черной магии". Программа работает вопреки здравому смыслу, совершенно непостижимому для окружающих!

А вот если переместить структуру ZERO в статическую область памяти (секцию данных), то размер, резервируемый под массив нулевой длины, сразу же сократиться до одного байта, поскольку выравнивать переменные в статической памяти нет никакой необходимости,

но в нашем распоряжении останется по меньшей мере один байт, который можно задействовать под производственные нужды ;)

трюк второй — динамические массивы на куче

Функции семейства malloc() обязаны корректно обрабатывать нулевой аргумент, возвращая валидный указатель на блок памяти нулевой длинны. Вот что говорит MSDN по этому поводу "If size is 0, malloc allocates a zero-length item in the heap and returns a valid pointer to that item" (*Если размер [выделяемой памяти] равен нулю, функция malloc выделяет блок памяти нулевой длины в куче и возвращает указатель на него*).

То есть, создавать массив нулевой длинны на куче мы можем без всяких извращений со структурами. Вот только обращаться к созданному массиву (по стандарту) никак не можем. Стандарт допускает только проверку указателя на нуль, сравнение двух указателей, освобождение памяти, ну и, естественно, реаллокацию. Однако, стандарт предполагает, а компилятор располагает.

Давайте выясним: сколько же всего в _действительности_ выделяется байт при создании массива нулевой длинны?

```
#define DATA_LEN       0       // нет данных

main()
{
        // создаем три массива нулевой длины
        char *p1=malloc(DATA_LEN);
        char *p2=malloc(DATA_LEN);
        char *p3=malloc(DATA_LEN);

        // создаем три массива длинной в один байт
        char *p4=malloc(1);
        char *p5=malloc(1);
        char *p6=malloc(1);

        // выводит указатель на созданные блоки на экран
        printf("0%Xh\n0%Xh\n0%Xh\n\n0%Xh\n0%Xh\n0%Xh\n", p1, p2, p3, p4, p5, p6);
}
```

Листинг 2 измерительный прибор, определяющий реальный размер массивов нулевой длинны

Откомпилировав программу с помощью Microsoft Visual C++ и запустив ее на выполнение, на мы получим следующий результат:

```
0300500h       ; \
03004F0h       ;  +- указатели на блоки нулевой длины
03004E0h       ; /

03004D0h       ; \
03004C0h       ;  +- указатель на блоки длиной в один байт
03004B0h       ; /
```

Листинг 3 реально выделяемый размер динамических массивов

Как видно, адреса выделяемых блоков планомерно уменьшаются на 10h байт, следовательно, каждый блок (состоящий из массива и служебных данных) занимает _намного_ больше, чем ничего. Более того, malloc(0) эквивалентно malloc(1). Определить размер актуальных данных динамического массива — несложно. Достаточно увеличивать аргумент, передаваемый malloc, до тех пор, пока разница между соседними указателями скачкообразно не увеличиться на некоторую величину.

Эксперимент показывает, что **минимальный размер выделяемого блока для Microsoft Visual C++ и 32-битных версий GCC составляет 10h байт**, то есть malloc(0) работает _точно_ _также_ как и malloc(0xF). Естественно, никаких гарантий, что остальные компиляторы поведут себя аналогичным образом, у нас нет и никогда не будет, поэтому, вылезать за границы отведенного блока по любому не стоит.

С другой стороны, выделив большое количество динамических массивов нулевого размера, не следует надеяться, что они не занимают драгоценной памяти и потому их можно не освобождать. Освобождать их нужно!!! Иначе память будет утекать со страшной скоростью!

трюк третий — оператор new

Практически все известные мне реализации Си++ компиляторов реализуют оператор new на основе malloc, поэтому, все, сказанное по отношению к malloc(0), справедливо и для new(0). Однако… кое-какие различия все-таки наблюдается и мне бы хотелось обратить на них читательское внимание.

Прежде всего откроем Стандарт (см. "C++ Programming Language, Second Edition" секция 5.3.3), где Дохлый Страус прямо так и пишет: "This implies that an operator new() can be called with the argument zero. In this case, a pointer to an object is returned. Repeated such calls return pointers to distinct objects" ("*...отсюда следует, что оператор new() может вызываться с нулевым аргументом и возвращать валидный указатель на объект. Последовательный вызов new(0) возвращает указатели на различные объекты*").

Дальше по тексту объясняется, что мы можем получить указатель на нулевой объект, сравнивать его с любым другим указателем, но вот обращение к объекту нулевой длинны стандартом… ну не то, чтобы запрещается, а отдается на откуп конкретным реализациям. Изучение исходных кодов RTL-библиотек различных компиляторов показывает, что new(0) в общем случае эквивалентно new(1) независимо от типа объекта.

Вот, например, фрагмент кода из GCC:

```
void* operator new(size_t size)       // реализация оператора new
{
        // если size равно нулю, принудительно устанавливаем размер в единицу
        if( size == 0 ) size = 1;
        …
        // продолжение функции
        …
}
```

Листинг 4 фрагмент кода из компилятора GCC, реализующий оператор new

Оператор new в свою очередь опирается на RTL-библиотеку, общую как для Си, так и для Си++, а потому оператор new(1) в большинстве случаев эквивалентен new(0xF), что наглядно подтверждает следующая программа:

```
main()
{
        // создаем символьный массив нулевой длины
        // (Стандартом это допускается)
        char *c = new char[0];

        // получаем указатель на созданный объект нулевой длины
        // (Стандартом это допускается)
        char *p = &c[0];

        // записываем в объект нулевой длинны число 0x69
        // (а вот этого Стандарт уже _не_ допускает!!!)
        *c=0x69;

        // проверяем успешность записи числа, выводя его на экран
        printf("0%Xh\n",*c);
}
```

Листинг 5 демонстрация создания объекта размеров в 1 байт с помощью new char[0]

Чтобы не быть голословным мыщъх приводит дизассемблерный фрагмент вышеупомянутой программы, откомпилированной Microsoft Visual C++ (__heap_alloc – служебная функция, на которую опирается оператор new) :

```
.text:00401B6F __heap_alloc   proc near             ; CODE XREF: __nh_malloc+B↑p
.text:00401B6F
.text:00401B6F arg_0          = dword ptr  8
.text:00401B6F
.text:00401B6F                push    esi
.text:00401B70                mov     esi, [esp+arg_0]       ; размер выделяемой памяти
.text:00401B74                cmp     esi, dword_406630      ; выделяем больше 1016байт
.text:00401B7A                ja      short loc_401B87       ; если да, то - прыжок
.text:00401B7C
.text:00401B7C                push    esi                    ; обрабатываем ситуацию
.text:00401B7D                call    ___sbh_alloc_block     ; с выделением <=1016байт
.text:00401B82                test    eax, eax               ; памяти
```

```
.text:00401B84                 pop     ecx
.text:00401B85                 jnz     short loc_401BA3       ; прыжок, если памяти нет
.text:00401B87
.text:00401B87 loc_401B87:                            ; CODE XREF: __heap_alloc+B↑j
.text:00401B87                 test    esi, esi               ; выделяем ноль байт?!
.text:00401B89                 jnz     short loc_401B8E       ; если не ноль, прыгаем
.text:00401B8B                 push    1                      ; если ноль, увеличиваем
.text:00401B8D                 pop     esi                    ; аргумент на единицу
.text:00401B8E
.text:00401B8E loc_401B8E:                            ; CODE XREF: __heap_alloc+1A↑j
.text:00401B8E                 add     esi, 0Fh               ; округляем размер блока
.text:00401B91                 and     esi, 0FFFFFFF0h        ; на 10h в большую сторону
.text:00401B94                 push    esi                    ; dwBytes
.text:00401B95                 push    0                      ; dwFlags
.text:00401B97                 push    hHeap                  ; hHeap
.text:00401B9D                 call    ds:HeapAlloc           ; выделяем блок памяти
.text:00401BA3
.text:00401BA3 loc_401BA3:                            ; CODE XREF: __heap_alloc+16↑j
.text:00401BA3                 pop     esi
.text:00401BA4                 retn
.text:00401BA4 __heap_alloc    endp
```

Листинг 6 дизассемблерный фрагмент функции __heap_alloc из Microsoft Visual C++, на которую опирается оператор new() и которая принудительно округляет выделяемый размер по границе 10h байт в большую строну, т.е. выделить менее 10h байт нам ни за что не удастся