Работа с динамической памятью в C

А. Г. Фенстер, http://info.fenster.name

28 января 2009 г.

Общая информация

Описанные ниже функции `malloc`, `calloc`, `realloc` и `free` необходимы для управления выделением и освобождением памяти в том случае, когда стандартного механизма управления памятью недостаточно. Они используются для возврата строк (`char *`) из функций, создания динамических массивов (т.е. массивов, размер которых неизвестен на этапе компиляции программы) и сложных структур в памяти (списки, деревья и т. п.).

Тип `void *`, с которым работают эти функции, является нетипизированным указателем, т. е. он может хранить адрес значения любого типа. При этом указатель типа `void *` нельзя разыменовывать: для обращения к значению необходимо привести его к указателю на некоторый тип (например, `int *` или `char *`). Здесь существует отличие между языками C и C++. В C не требуется явного приведения типа при присваивании типизированному указателю значения нетипизированного, т. е. присваивание во второй строке будет откомпилировано без ошибок:

```
void *p = ...;
int *q = p;
```

В то же время в C++ в данном случае необходимо явное приведение типа:

```
void *p = ...;
int *q = (int *)p; // либо q = static_cast<int *>(p);
```

Гарантируется, что ни одна реальная ячейка памяти не имеет нулевого адреса. Это позволяет использовать `0` (`NULL`) в качестве возвращаемого значения, указывающего на ошибку (например, недостаточно свободной памяти). Символ `NULL` в языке C обычно объявлен одним из двух следующих способов:

```
#define NULL ((void *)0)
#define NULL 0
```

и использование `NULL` вместо 0 для обозначения недействительного указателя является, пожалуй, лишь вопросом стиля.

Тип `size_t` используется для хранения размеров выделяемых блоков памяти и на самом деле является беззнаковым целым числом. Так, в ОС Linux на архитектуре x86 его размер равен 4 байтам и он совпадает с типом `unsigned int`. Часто бывает необходимо определить размер конкретного типа, для этой цели в C используется оператор `sizeof` (например, `sizeof(int)`).

Функции работы с динамической памятью

`void *malloc(size_t size);`
Действие: выделяет в динамической памяти блок размером `size` байт. Выделенная память не инициализирована и заполнена произвольными значениями!
Возвращает нетипизированный указатель на начало блока.
В случае ошибки возвращает `NULL`. Хорошим стилем является проверка возвращаемого функцией `malloc` значения перед его использованием (хотя в небольших программах единственной адекватной реакцией на такую ошибку будет завершение работы программы).

`void *calloc(size_t nmemb, size_t size);`
Действие: выделяет в динамической памяти сплошной массив из `nmemb` элементов, по `size` байт каждый. Выделенная память инициализируется нулями.
Возвращает нетипизированный указатель на начало массива.
В случае ошибки возвращает `NULL`. Аналогично функции `malloc`, возвращаемое значение `calloc` желательно проверять.

`void *realloc(void *ptr, size_t size);`
Действие: изменяет размер блока памяти, выделенного ранее функциями `malloc`, `calloc` или `realloc`, делая его равным `size` байт. Функция `realloc` может действовать одним из трёх способов:

1. если `ptr == NULL`, функция работает аналогично функции `malloc` с параметром `size`.
2. если запрос на изменение размера возможно выполнить, функция возвращает новый адрес блока (он может быть равным предыдущему или отличаться от него). Если блок был перемещён в новое место, то данные также были скопированы туда. После выполнения функции по старому адресу (`ptr`) обращаться нельзя.
3. в случае невозможности выполнить запрос на изменение размера функция возвращает `NULL`, при этом данные по адресу `ptr` остаются на месте и к ним можно продолжать обращаться.

`void free(void *ptr);`
Действие: освобождает блок памяти, на начало которого указывает `ptr`. Этот блок должен быть выделен ранее при помощи `malloc`, `calloc` или `realloc`. Если в небольших программах, работающих в течение короткого времени, освобождение выделенной памяти является лишь вопросом хорошего стиля, то в программах большего размера необходимо следить за всей выделенной памятью, чтобы не допустить «утечки памяти» (когда программа с течением времени потребляет всё больше и больше памяти, при этом теряя ранее выделенную память).

Примеры

Создание и освобождение динамического массива

```
int *p = malloc(N * sizeof(int));
        // либо p = calloc(N, sizeof(int));
if (!p) // p == NULL
{
    perror("malloc"); // сообщение об ошибке
```

```
    return;
}
// с p можно работать как с обычным массивом
for (i = 0; i < N; i++)
    p[i] = i;
// освобождение памяти:
free(p);
```

Изменение размера выделенной области

```
int *p = malloc(N * sizeof(int));
...
int *q = realloc(p, M * sizeof(int));
if (!q)
{
    perror("realloc");
    // действия при ошибке
}
else
{
    p = q;
}
...
free(p);
```