# КОНСТРУИРОВАНИЕ ПРОГРАММ

**Лекция № 15 — Перегрузка и шаблоны**

**Преподаватель: Поденок Леонид Петрович, 505a-5**

**+375 17 293 8039 (505a-5)**

**+375 17 320 7402 (ОИПИ НАНБ)**

**prep@lsi.bas-net.by**

**ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/**

**Кафедра ЭВМ, 2021**

2021.11.24

# Оглавление

## Динамическое выделение памяти в C — <stdlib.h>

**malloc**, **free**, **calloc**, **realloc** — распределяют и освобождают динамическую память

```
#include <stdlib.h>

void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
void *calloc(size_t nmemb, size_t size);
```

**malloc()**

Функция **malloc()** распределяет **size** байтов и возвращает указатель на распределённую память. Память при этом не инициализируется.

Если значение **size** равно 0, то **malloc()** возвращает или **NULL**, или уникальный указатель, который можно без опасений передавать **free()**.

**free()**

Функция **free()** освобождает место в памяти, указанное в **ptr**, которое должно быть получено ранее вызовом функции **malloc()**, **calloc()** или **realloc()**.

В противном случае (или если вызов **free(ptr)** уже выполнялся) дальнейшее поведение не определено.

Если значение **ptr** равно **NULL**, то не выполняется никаких действий.

Функция **free()** ничего не возвращает.

**realloc()**

Функция **realloc()** меняет размер блока памяти, на который указывает **ptr**, на размер, равный **size** байт.

Содержимое памяти от начала области в пределах наименьшего из старого и нового размеров не будет изменено.

Если новый размер больше старого, то добавленная память не будет инициализирована.

Если значение **ptr** равно **NULL**, то вызов эквивалентен **malloc(size)** для всех значений **size**.

Если значение **size** равно нулю и **ptr** не равно **NULL**, то вызов эквивалентен **free(ptr)**.

Функция **realloc()** возвращает указатель на новую распределённую память, выровненную должным образом для любого встроенного типа. Возвращаемый указатель может отличаться от **ptr**, или равняться **NULL**, если запрос завершился с ошибкой.

Если значение **size** было равно нулю, то возвращается либо **NULL**, либо указатель, который может быть передан **free()**.

Если **realloc()** завершилась с ошибкой, то начальный блок памяти остаётся нетронутым — он ни освобождается, ни или

**calloc()**

Функция **calloc()** распределяет память для массива размером **nmemb** элементов по **size** байтов каждый и возвращает указатель на распределённую память. Данные в выделенной памяти при этом обнуляются. Если значение **nmemb** или **size** равно **0**, то **calloc()** возвращает или **NULL**, или уникальный указатель, который можно без опасений передавать **free()**.

Функции **malloc()** и **calloc()** возвращают указатель на распределённую память, выровненную должным образом для любого **встроенного типа**.

При ошибке возвращается **NULL**. Значение **NULL** также может быть получено при успешной работе вызова **malloc()**, если значение **size** равно нулю, или **calloc()** — если значение **nmemb** или **size** равно нулю.

**Ошибки**

Функции **calloc()**, **malloc()**, **realloc()** могут завершаться со следующей ошибкой:

**ENOMEM** — не хватает памяти. В этом случае скорее всего приложением достигнут лимит **RLIMIT_AS** или **RLIMIT_DATA**, описанный в **getrlimit(2)**.

# Динамическое выделение памяти в C++

С целью удовлетворения потребности в распределения памяти в язык C++ интегрировали операторы **new** и **delete**.

## Операторы new и new[ ]

Динамическая память выделяется с помощью оператора **new**. За символом **new** следует спецификатор типа данных, и, если требуется последовательность из более чем одного элемента, их число в квадратных скобках **[ ]**. В результате возвращается указатель на начало нового выделенного блока памяти.

Синтаксис оператора:

*pointer* **= new T**
*pointer* **= new T [** *number_of_elements* **]**

Первое выражение используется для выделения памяти под размещение одного элемента типа **T**.

Второе используется для выделения блока (массива) элементов типа **T**, где целочисленное значение *number_of_elements* представляет их количество.

```
int *foo;           // объявление указателя
foo = new int[5];   // инициализация
```

или

```
int *foo = new int[5]; // все сразу
```

В этом случае система динамически распределяет пространство для пяти элементов типа **int** и возвращает указатель на первый элемент последовательности, который присваивается указателю **foo**. После этого **foo** указывает на допустимый блок памяти с пространством на пять элементов типа **int**.

Здесь **foo** является указателем, и, таким образом, к первому элементу, на который указывает foo, можно получить доступ с помощью выражения **foo[0]** или с помощью выражения ***foo** (оба являются эквивалентными). Ко второму элементу можно получить доступ либо с помощью **foo[1]**, либо ***(foo + 1)**, и так далее ...

Существует разница между объявлением обычного массива и выделением динамической памяти для блока памяти с использованием **new**. Наиболее важным отличием является то, что размер регулярного (с внешним или внутренним типом связывания) массива должен быть постоянным выражением, и, следовательно, его размер должен быть определен в момент компилирования программы, тогда как динамическое распределение памяти, выполняемое **new**, позволяет выделять память во время выполнения, используя любое значение в качестве размера.

## Обработка ошибок выделения

Динамическая память, запрошенная данной программой, выделяется системой из кучи (heap). Однако память компьютера является ограниченным ресурсом и может быть исчерпана. Таким образом, нет никаких гарантий, что все запросы на выделение памяти с использованием оператора **new** будут предоставлены системой.

C++ предоставляет два стандартных механизма для проверки успешного распределения — *обработка исключений* и *нулевой указатель*.

**Обработка исключений** — при использовании этого метода при сбое выделения генерируется исключение типа **bad_alloc**.

Исключения — это мощный функционал C++, который будет объяснен позже.

Если это исключение выдается, и оно не обрабатывается определенным обработчиком, выполнение программы прекращается.

```
#include <iostream>     // std::cout
#include <new>          // std::bad_alloc

int main () {

  try {
    int* myarray= new int[10000000000]; //
  } catch (std::bad_alloc& ba) {
    std::cerr << "bad_alloc caught: " << ba.what() << '\n';
} }
```

**Вывод**

```
bad_alloc caught: bad allocation
```

Метод обработки исключений является методом, используемым оператором **new** по умолчанию в выражениях типа:

```
foo = new int[5];  // if allocation fails, an exception is thrown
```

**Нулевой указатель** — этот метод известен как **nothrow**, и когда он используется, происходит следующее — когда происходит сбой выделения памяти, вместо того, чтобы вызывать (***throw***) исключение **bad_alloc** или завершать программу, **new** возвращает нулевой указатель, и программа продолжает свое выполнение как обычно.

Данный метод указывается с помощью специального объекта **nothrow**, который объявлен в заголовке **<new>**, в качестве аргумента для **new**:

```
foo = new (nothrow) int[5];
```

В этом случае, если выделение этого блока памяти не было выполнено, сбой можно обнаружить, проверив, является ли **foo** нулевым указателем:

```
int *foo;
...
foo = new (nothrow) int [5];
if (foo == nullptr) {              // (nullptr == foo)
    // ошибка выделения памяти.
}
```

Считается, что метод **nothrow** производит менее эффективный код, чем исключение, поскольку он подразумевает явную проверку значения указателя, возвращаемого после каждого запроса на выделение памяти. Поэтому механизм исключения обычно считается более предпочтительным.  Тем не менее механизм **nothrow** гораздо проще, а проверка так или иначе выполняется в любом случае.

Всегда существует вероятность того, что будет запрошено памяти больше, нежели система может выделить.

Хорошей практикой считается, что программы всегда должны обрабатывать сбои при выделении памяти, либо проверяя значение указателя (если **nothrow**), либо перехватывая соответствующее исключение.

Библиотечные функции **malloc**, **calloc**, **realloc** и **free**, определенные в заголовке **<cstdlib>** (**<stdlib.h>**) также доступны в C++ и могут также использоваться для выделения и освобождения динамической памяти.

Следует помнить, что блоки памяти, выделенные этими функциями, не обязательно совместимы с теми, которые возвращаются **new**, поэтому их не следует смешивать — каждый из них должен обрабатываться со своим набором функций или операторов.

## Операторы **delete** и **delete[]**

В большинстве случаев память, выделенная динамически, необходима программе только в течение определенных периодов времени и, когда она больше не нужна, имеет смысл ее освободить, чтобы память снова стала доступной для других запросов. Это цель оператора **delete** с синтаксисом:

**delete** *pointer*;

**delete[]** *pointer*;

Первая инструкция освобождает память одного элемента, выделенного с помощью **new**, а вторая освобождает память, выделенную для массивов элементов с использованием **new** и размером в квадратных скобках **[]**.

Значение, передаваемое в качестве аргумента для удаления, должно быть либо указателем на блок памяти, ранее выделенный с помощью **new**, либо нулевым указателем (в случае нулевого указателя удаление не дает никакого эффекта).