Программирование на языке C++
Лекция 2

Динамическая память

Александр Смаль

Зачем нужна динамическая память?

- Стек программы ограничен. Он не предназначен для хранения больших объемов данных.

```
// Не умещается на стек
double m[10000000] = {}; // 80 Mb
```

- Время жизни локальных переменных ограничено временем работы функции.
- Динамическая память выделяется в сегменте данных.
- Структура, отвечающая за выделение дополнительной памяти, называется **кучей** (не нужно путать с одноимённой структурой данных).
- Выделение и освобождение памяти *управляется вручную*.

Выделение памяти в стиле C

- Стандартная библиотека `cstdlib` предоставляет четыре функции для управления памятью:

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

- `size_t` — специальный целочисленный беззнаковый тип, может вместить в себя размер любого типа в байтах.
- Тип `size_t` используется для указания размеров типов данных, для индексации массивов и пр.
- `void *` — это указатель на нетипизированную память (раньше для этого использовалось `char *`).

Выделение памяти в стиле C

- Функции для управления памятью в стиле C:

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

- `malloc` — выделяет область памяти размера $\geq$ `size`. Данные не инициализируются.
- `calloc` — выделяет массив из `nmemb` размера `size`. Данные инициализируются нулём.
- `realloc` — изменяет размер области памяти по указателю `ptr` на `size` (если возможно, то это делается на месте).
- `free` — освобождает область памяти, ранее выделенную одной из функций `malloc`/`calloc`/`realloc`.

Выделение памяти в стиле C

- Для указания размера типа используется оператор `sizeof`.

```
// создание массива из 1000 int
int * m = (int *)malloc(1000 * sizeof(int));
m[10] = 10;

// изменение размера массива до 2000
m = (int *)realloc(m, 2000 * sizeof(int));

// освобождение массива
free(m);

// создание массива нулей
m = (int *)calloc(3000, sizeof(int));

free(m);
m = 0;
```

Выделение памяти в стиле C++

- Язык C++ предоставляет два набора операторов для выделения памяти:
  1. `new` и `delete` — для одиночных значений,
  2. `new []` и `delete []` — для массивов.
- Версия оператора `delete` должна соответствовать версии оператора `new`.

```
// выделение памяти под один int со значением 5
int * m = new int(5);
delete m; // освобождение памяти

// создание массива значений типа int
m = new int[1000];
delete [] m; // освобождение памяти
```

Типичные проблемы при работе с памятью

- Проблемы производительности: создание переменной на стеке намного “дешевле” выделения для неё динамической памяти.
- Проблема фрагментации: выделение большого количества небольших сегментов способствует фрагментации памяти.
- Утечки памяти:

```
// создание массива из 1000 int
int * m = new int[1000];

// создание массива из 2000 int
m = new int[2000]; // утечка памяти

// Не вызван delete [] m, утечка памяти
```

Типичные проблемы при работе с памятью

- Неправильное освобождение памяти.

```
int * m1 = new int[1000];
delete m1; // должно быть delete [] m1

int * p = new int(0);
free(p); // совмещение функций C++ и C

int * q1 = (int *)malloc(sizeof(int));
free(q1);
free(q1); // двойное удаление

int * q2 = (int *)malloc(sizeof(int));
free(q2);
q2 = 0;    // обнуляем указатель
free(q2); // правильно работает для q2 = 0
```