Лекция 4. Динамическая память

Александр Смаль

CS центр
12 сентября 2017
Санкт-Петербург

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

- `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` элементов 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 нулей
m = new int[1000](); // () означает обнуление
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
```

Многомерные встроенные массивы

- C++ позволяет определять многомерные массивы:

```
int m2d[2][3] = { {1, 2, 3}, {4, 5, 6} };
for( size_t i = 0; i != 2; ++i ) {
    for( size_t j = 0; j != 3; ++j ) {
        cout << m2d[i][j] << ' ';
    }
    cout << endl;
}
```

- Элементы `m2d` располагаются в памяти “по строчкам”.
- Размерность массивов может быть любой, но на практике редко используют массивы размерности > 4.

```
int m4d[2][3][4][5] = {};
```

Динамические массивы

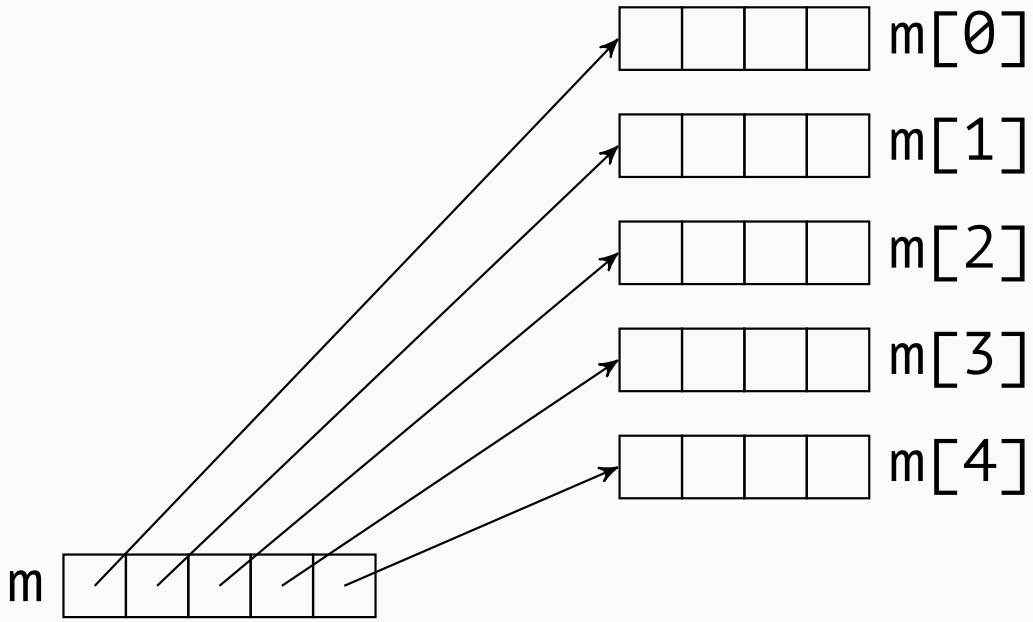
- Для выделения одномерных динамических массивов обычно используется оператор `new []`.

```
int * m1d = new int[100];
```

- Какой тип должен быть у указателя на двумерный динамический массив?
  - Пусть `m` — указатель на двумерный массив типа `int`.
  - Значит `m[i][j]` имеет тип `int` (точнее `int &`).
  - `m[i][j]` ⇔ `*(m[i] + j)`, т.е. тип `m[i]` — `int *`.
  - аналогично, `m[i]` ⇔ `*(m + i)`, т.е. тип `m` — `int **`.
- Чему соответствует значение `m[i]`?
  Это адрес строки с номером `i`.
- Чему соответствует значение `m`?
  Это адрес массива с указателями на строки.

Двумерные массивы

Давайте рассмотрим создание массива 5×4.




```
int ** m = new int * [5];
for (size_t i = 0; i != 5; ++i)
    m[i] = new int[4];
```

Двумерные массивы

Выделение и освобождение двумерного массива размера $a \times b$.

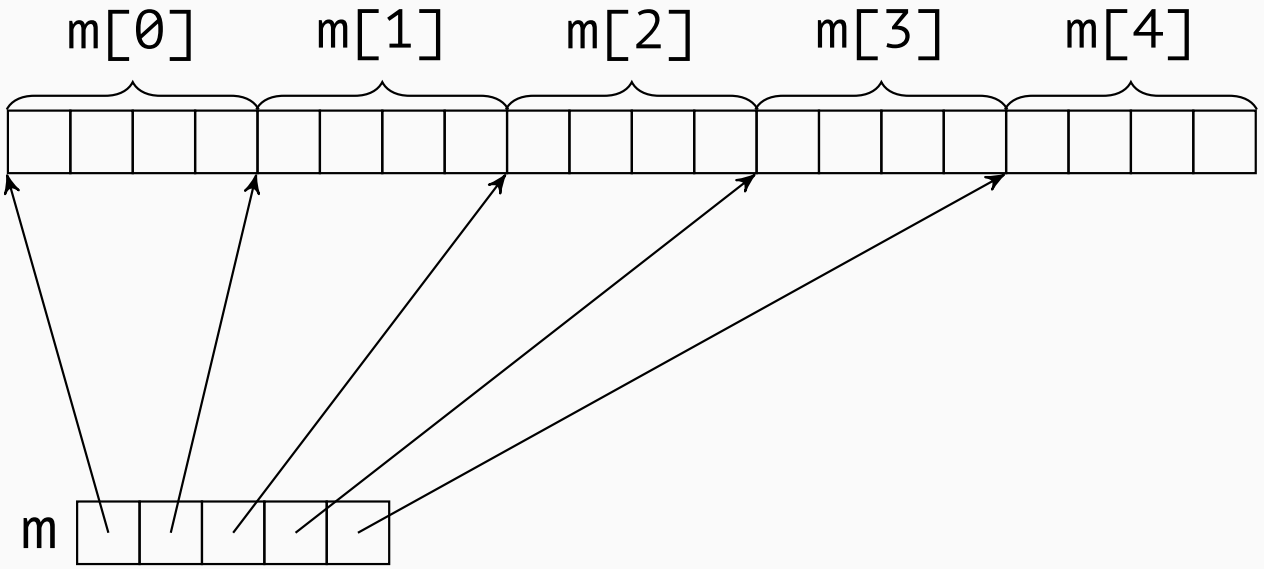
```
int ** create_array2d(size_t a, size_t b) {
    int ** m = new int *[a];
    for (size_t i = 0; i != a; ++i)
        m[i] = new int[b];
    return m;
}

void free_array2d(int ** m, size_t a, size_t b) {
    for (size_t i = 0; i != a; ++i)
        delete [] m[i];
    delete [] m;
}
```

При создании массива оператор `new` вызывается $(a + 1)$ раз.

Двумерные массивы: эффективная схема

Рассмотрим эффективное создание массива 5×4.




```
int ** m = new int * [5];
m[0] = new int[5 * 4];
for (size_t i = 1; i != 5; ++i)
    m[i] = m[i - 1] + 4;
```

Двумерные массивы: эффективная схема

Эффективное выделение и освобождение двумерного массива размера $a \times b$.

```
int ** create_array2d(size_t a, size_t b) {
    int ** m = new int *[a];
    m[0] = new int[a * b];
    for (size_t i = 1; i != a; ++i)
        m[i] = m[i - 1] + b;
    return m;
}

void free_array2d(int ** m, size_t a, size_t b) {
    delete [] m[0];
    delete [] m;
}
```

При создании массива оператор new вызывается 2 раза.

Строковые литералы

- Строки — это массивы символов типа `char`, заканчивающиеся нулевым символом.

```
// массив 'H', 'e', 'l', 'l', 'o', '\0'
char s[] = "Hello";
```

- Строки могут содержать управляющие последовательности:
  1. `\n` — перевод строки,
  2. `\t` — символ табуляции,
  3. `\\` — символ '\',
  4. `\"` — символ '"',
  5. `\0` — нулевой символ.

```
cout << "List:\n\t- C,\n\t- C++.\n";
```

Работа со строками в стиле C

- Библиотека `cstring` предлагает множество функций для работы со строками (`char *`).

```
char s1[100] = "Hello";
cout << strlen(s1) << endl; // 5

char s2[] = ", world!";
strcat(s1, s2);

char s3[6] = {72, 101, 108, 108, 111};
if (strcmp(s1, s3) == 0)
    cout << "s1 == s3" << endl;
```

- Работа со строками в стиле C предполагает кропотливую работу с ручным выделением памяти.

Работа со строками в стиле C++

Библиотека `string` предлагает обёртку над строками, которая позволяет упростить все операции со строками.

```
#include <string>
using namespace std;

int main() {
    string s1 = "Hello";
    cout << s1.size() << endl; // 5

    string s2 = ", world!";
    s1 = s1 + s2;
    if (s1 == s2)
        cout << "s1 == s2" << endl;
    return 0;
}
```

Ввод-вывод в стиле C

- Библиотека `cstdio` предлагает функции для работы со стандартным вводом-выводом.
- Для вывода используется функция `printf`:

```
#include <cstdio>

int main() {
    int h = 20, m = 14;
    printf("Time: %d:%d\n", h, m);
    printf("It's %.2f hours to midnight\n",
           ((24 - h) * 60.0 - m) / 60);
    return 0;
}
```

Ввод-вывод в стиле C

- Библиотека `cstdio` предлагает функции для работы со стандартным вводом-выводом.
- Для ввода используется функция `scanf`:

```
#include <cstdio>

int main() {
    int a = 0, b = 0;
    printf("Enter a and b: ");
    scanf("%d %d", &a, &b);
    printf("a + b = %d\n", (a + b));
    return 0;
}
```

- Ввод-вывод в стиле C достаточно сложен и небезопасен (типы аргументов не проверяются).

Ввод-вывод в стиле C++

- В C++ ввод-вывод реализуется через библиотеку `iostream`.

```
#include <string>
#include <iostream>
using namespace std;

int main() {
    string name;
    cout << "Enter your name: ";
    cin >> name; // считывается слово
    cout << "Hi, " << name << endl;

    return 0;
}
```

- Реализация ввода-вывода в стиле C++ типобезопасна.

Работа с файлами в стиле C++

- Библиотека `fstream` обеспечивает работу с файлами.

```
#include <string>
#include <fstream>
using namespace std;

int main() {
    string name;
    ifstream input("input.txt");
    input >> name;

    ofstream output("output.txt");
    output << "Hi, " << name << endl;
    return 0;
}
```

- Файлы закроются при выходе из функции.