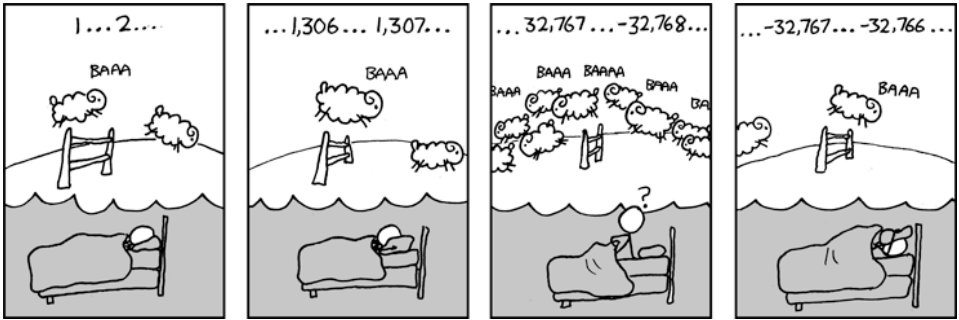
Основы программирования

Лекция № 2, 2 марта 2017 г.



http://xkcd.com/571

Знакомство

Я — Владимир Владимирович Парфиненко,

- бакалавр физики (ФФ), магистр математики (ММФ),
- профессиональный программист (Excelsior),
- регулярно чему-то учу (ФФ, АФТИ, ЛШ ФМШ).

Контакт: vladimir.parfinenko@gmail.com

Целочисленные типы данных

Двоичная система счисления

Самый простой метод записи чисел, использующий только две цифры: 0 и 1.

С помощью n *позиций* можно записать 2^n чисел.

$$\texttt{000}_2 = 0, \qquad \texttt{100}_2 = 4,$$
$$\texttt{001}_2 = 1, \qquad \texttt{101}_2 = 5,$$
$$\texttt{010}_2 = 2, \qquad \texttt{110}_2 = 6,$$
$$\texttt{011}_2 = 3, \qquad \texttt{111}_2 = 7.$$

Числа в памяти компьютера

1 *байт* состоит из 8 *бит* и может кодировать $2^8 = 256$ различных чисел.

Например, число 337 кодируется минимум 2 байтами:

$$\underbrace{00000001}_{\text{биты 15...8}}\underbrace{01010001}_{\text{биты 7...0}}$$

- 1 байт (8 бит) кодирует 256 чисел,
- 2 байта (16 бит) кодируют 65 536 чисел,
- 4 байта (32 бита) кодируют 4 294 967 296 чисел,
- 8 байт (64 бита) кодируют 18 446 744 073 709 551 616 чисел.

Целочисленные типы в C

Тип	Размер, бит	Минимум	Максимум
`unsigned char`	8	0	255
`unsigned short`	16	0	65 535
`unsigned int`	32	0	4 294 967 295
`unsigned long long`	64	0	$18{,}4 \cdot 10^{18}$
`signed char`	8	−128	127
`short`	16	−32 768	32 767
`int`	32	−2 147 483 648	2 147 483 647
`long long`	64	$-9{,}2 \cdot 10^{18}$	$9{,}2 \cdot 10^{18}$

Сложение целых чисел с переполнением

Рассмотрим сложение двух 8-битных беззнаковых чисел

$$250 + 9 = \mathtt{1111\,1010}_2 + \mathtt{0000\,1001}_2 = \ldots$$

$$\begin{array}{r} \mathtt{1111\,1010} \\ + \quad \mathtt{0000\,1001} \\ \hline 1\,\underbrace{\mathtt{0000\,0011}}_{\text{8 бит}} \end{array}$$

С точки зрения компьютера: $250 + 9 = \mathtt{0000\,0011}_2 = 3$.

Виды переполнения целых чисел

Беззнаковые 8-битные:

- $255 + 1 = 0$,
- $0 - 1 = 255$.

Знаковые 8-битные:

- $127 + 1 = -128$,
- $-128 - 1 = 127$.

Epic Fails

- 13 июня 2009 г. — Twitter: порядковый номер твитов переполнил 32-битное знаковое целое.
- 22 сентября 2009 г. — Twitter: порядковый номер твитов переполнил 32-битное беззнаковое целое.
- 9 февраля 2013 г. — OpenStreetMap: порядковый номер точек на карте переполнил 32-битное знаковое целое.
- 1 декабря 2014 г. — YouTube: количество просмотров одного видео переполнило 32-битное знаковое целое.
- 20 января 2017 г. — Code.org: идентификатор студенческих работ переполнил 32-битное беззнаковое целое.

Вещественные типы данных

Числа с плавающей точкой

$$x = m \cdot b^e,$$

где:

- m — *мантисса* (значащая часть),
- b — *основание степени* (обычно 2 или 10),
- e — *экспонента* (порядок).

Числа с плавающей точкой в C

Тип	Мин. абс. значение	Макс. абс. значение	Точность, дес. знаков
`float`	$1{,}18 \cdot 10^{-38}$	$3{,}40 \cdot 10^{38}$	≈ 7
`double`	$2{,}23 \cdot 10^{-308}$	$1{,}80 \cdot 10^{308}$	≈ 16

Особенные числа

Числа ± 0, $\pm\infty$

$$\log(0) = -\infty,$$
$$1/(+0) = +\infty,$$
$$1/(-0) = -\infty.$$

Не-числа (NaN, Not a Number):

$$NaN = \sqrt{-1}, 0/0, 0 \cdot \infty, \infty/\infty, \infty - \infty,$$

Проблемы с точностью

- Округление: $\mathrm{tg}(\pi) \neq 0, \mathrm{tg}(\pi/2) \neq \infty$.
- Накопление ошибок: $a + (b + c) \neq (a + b) + c$.
- Потеря точности: $a - b$, если $a \approx b$.
- Потеря точности: $a + b$, если $a \gg b$ или $a \ll b$.
- Сравнение чисел.

Массивы

Объявление массива

Массив — упорядоченный набор элементов одного типа.

Объявление массива `arr`, состоящего из N элементов произвольного типа T:

```
T arr[N];
```

где N — константа времени компиляции.

Пример объявления и инициализации массива:

```
int xs[4];             // {?, ?, ?, ?}
int ys[]  = {20, 10}; // {20, 10}
int zs[4] = {20, 10}; // {20, 10, 0, 0}
```

Доступ к элементам массива

`T arr[N]`:

Чтение элемента из массива по индексу `i`:

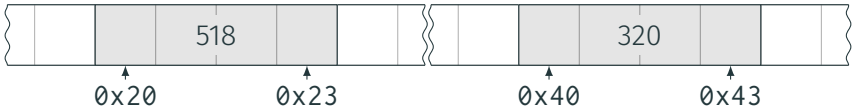
```
T value = arr[i];
```

Запись элемента в массив по индексу `i`:

```
arr[i] = new_value;
```

Указатели

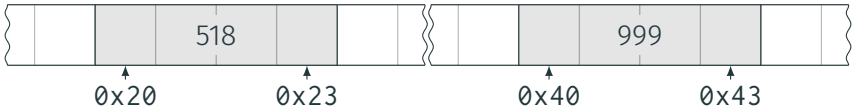
Указатель как адрес




```c
int x = 518;
int y = 320;
```

```c
int* ptr;
// & - взятие адреса
// * - разыменование указателя
ptr = &x;
printf("%p %d\n", ptr, *ptr); // 0x20 518
ptr = &y;
printf("%p %d\n", ptr, *ptr); // 0x40 320
```

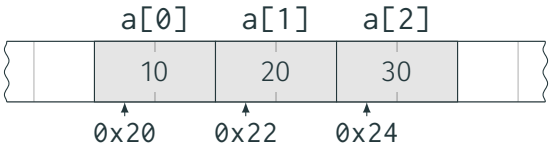
Указатель как адрес




```c
int x = 518;
int y = 320;
```

```c
int* ptr;
// & - взятие адреса
// * - разыменование указателя
ptr = &x;
printf("%p %d\n", ptr, *ptr); // 0x20 518
ptr = &y;
printf("%p %d\n", ptr, *ptr); // 0x40 320
*ptr = 999;
printf("%d %d\n", x, y); // 518 999
```

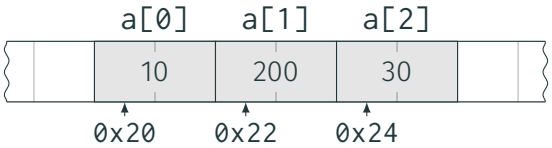
Массивы и указатели




```
short a[3] = {10, 20, 30};

short* p = &(a[0]); // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

Массивы и указатели

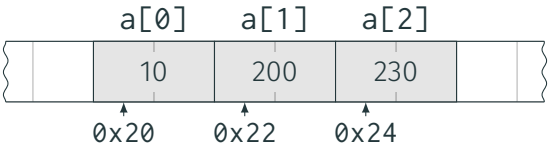



```
short a[3] = {10, 20, 30};
```

```
*p1 = 200;
```

```
short* p = &(a[0]); // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

Массивы и указатели

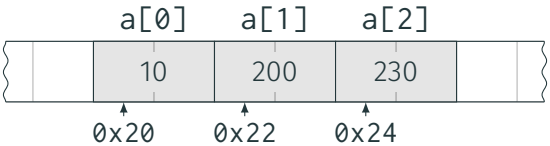



```
short a[3] = {10, 20, 30};

short* p = &(a[0]); // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

```
*p1 = 200;
*p2 = *p1 + *p2;
```

Массивы и указатели

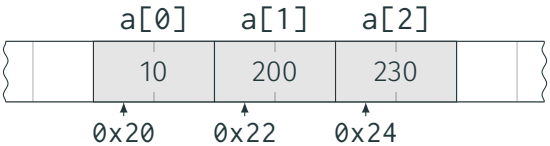



```
short a[3] = {10, 20, 30};

short* p = &(a[0]); // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

```
*(p+1) = 200;
*(p+2) = *(p+1) + *(p+2);
```

Массивы и указатели

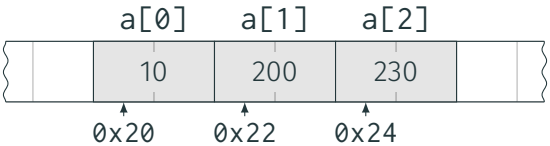



```
short a[3] = {10, 20, 30};

short* p = &(a[0]); // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

```
p[1] = 200;
p[2] = p[1] + p[2];
// *(x + y) == x[y]
```

Массивы и указатели

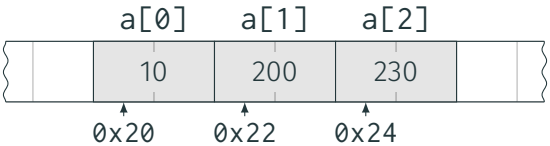



```
short a[3] = {10, 20, 30};

short* p = &(*(a)); // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

```
p[1] = 200;
p[2] = p[1] + p[2];
// *(x + y) == x[y]
```

Массивы и указатели




```
short a[3] = {10, 20, 30};

short* p = a; // 0x20
short* p1 = p + 1;  // 0x22
short* p2 = p + 2;  // 0x24
```

```
p[1] = 200;
p[2] = p[1] + p[2];
// *(x + y) == x[y]
```

Остальное не влезло в пару...

Указатель, никуда не указывающий

NULL — специальное константное значение, символизирующее, что указатель не указывает ни на какой элемент памяти. Объявлено в заголовочном файле `stdlib.h`.

```
int* ptr = NULL;
int value = *ptr; // run time error
*ptr = 37; // run time error
```

Разные названия, но суть одна (segmentation fault, segfault, access violation, «Программа выполнила недопустимую операцию...», ...).

Указатель, указывающий хоть куда

`void*` — специальный тип указателя, который может указывать на любой адрес в памяти. Может быть приведен к любому другому типу указателей и обратно.

```
double x = 37;
double* px = &x;
void* p = px;
int* py = p;
```

Минутка философии: «Какова природа void?» — спросил учитель, ...

Динамическая память

Преимущества динамической памяти

- Выделяется и освобождается динамически по запросу программы.
- Размер задается динамически.

Выделение блока памяти

```
void* malloc(size_t size);
```

Функция выделяет блок памяти размером `size` байт и возвращает указатель на начало блока. В случае, если память выделить не получилось, возвращает `NULL`. Объявлена в заголовочном файле `stdlib.h`.

Освобождение блока памяти

```
void free(void* ptr);
```

Функция освобождает блок памяти. Если `ptr` равен `NULL`, ничего не делает. Объявлена в заголовочном файле `stdlib.h`.

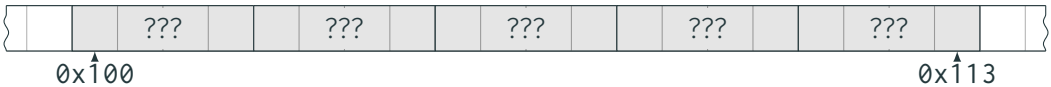
После вызова значение указателя `ptr` остается прежним, но разыменовывать его нельзя.

Неиспользуемую память нужно обязательно освобождать, иначе рано или поздно она может кончиться (утечка памяти).

Пример: массив динамического размера

```
int n = read_number(); // 5
int* p; // ???
```

Пример: массив динамического размера

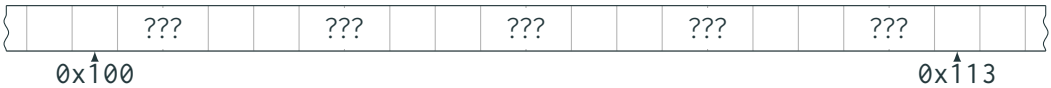



```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
```

Пример: массив динамического размера


```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
p[0] = p[n/2] = p[n-1] = 37;
```

Пример: массив динамического размера




```
int n = read_number(); // 5
int* p; // 0x100
p = malloc(n * sizeof(int));
if (p == NULL) { /* error */ }
p[0] = p[n/2] = p[n-1] = 37;
free(p);
```

Конец второй лекции