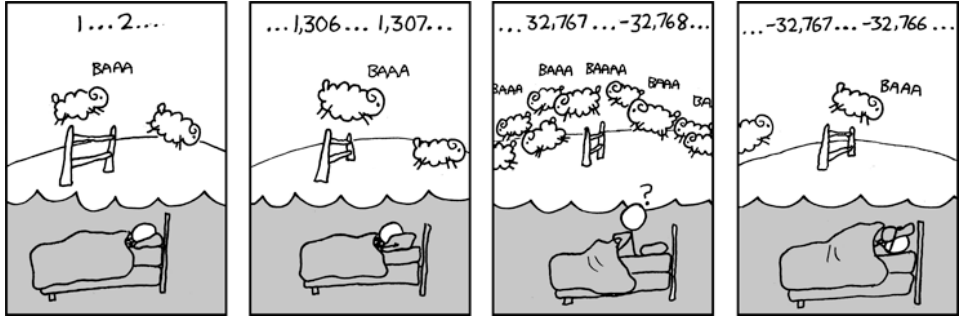
Основы программирования

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

Заметим, что с помощью двух ламп можно представить числа от 0 до 3.

Двоичная система счисления

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

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

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

Компьютеру легко работать с такими числами.

Индекс x_{10} будем опускать.

Правила перевода из двоичной в десятичную и обратно выходит за рамки данной лекции.

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:

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

- 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.

Байт - минимальная ячейка памяти, которую компьютер может читать/писать.

$18 \cdot 10^{18}$ = восемнадцать квинтиллионов.

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	$-9{,}2 \cdot 10^{18}$	$9{,}2 \cdot 10^{18}$

Эти значения могут варьироваться в зависимости от платформы и компилятора.

Знаковость char не стандартизирована.

Знак хранится в тех же самых 8/16/... битах. Детали представления отрицательных чисел выходят за рамки данной лекции.

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

То есть компьютер выполняет сложение и вычитание по модулю, соответствующему размеру числа, т.е. отбрасывает старшую часть.

Виды переполнения целых чисел

Беззнаковые 8-битные:

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

Знаковые 8-битные:

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

Переполнение беззнаковых чисел в C стандартизовано.

Переполнение знаковых чисел в C — неопределенное поведение.

Про переполнение надо знать, но лучше на него не полагаться.

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.

В абсолютном большинстве случаев хватит 32-битных чисел.

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).

Трудности перевода: в России разделителем служит запятая.

Очень привычная для физиков форма записи вещественный чисел.

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

Точность binary32: $\log_{10}(2^{23} + 1) \approx 7{,}225$,
точность binary64: $\log_{10}(2^{52} + 1) \approx 15{,}955$.

Мин. и макс. значения даны для нормализованных чисел.

Проще всегда использовать double.

Особенные числа

Числа ± 0, $\pm\infty$

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

Не-числа (NaN, Not a Number):

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

Есть еще денормализованные числа, но это выходит за рамки данной лекции.

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.

$\frac{f(a)-f(b)}{a-b}$ — в числителе значащими становятся младшие, неточные разряды.

Вещественные числа нужно сравнивать приблизительно.
Подробности выходят за рамки этой лекции.

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}
```


Мотивация: для задания квадратного уравнения нужно 3 переменные: a, b, c. А что делать с уравнением 10-ой степени? Правильно, нужен массив!

Без инициализации массив, объявленный в функции, содержит мусор.

Элементы, для которых нет явно заданного значения, инициализируются нулями.

Размер массива может быть вычислен автоматически, что может быть довольно удобным.

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`:

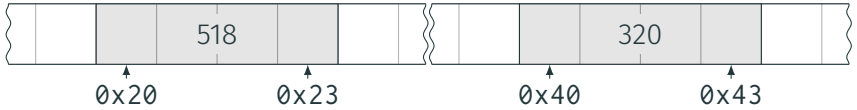
```
arr[i] = new_value;
```

Индексы в массиве от 0 до N-1.

Доступ по недопустимому индексу (-1, N, 2*N) — неопределенное поведение.

Указатели

Указатель как адрес




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

```
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
```

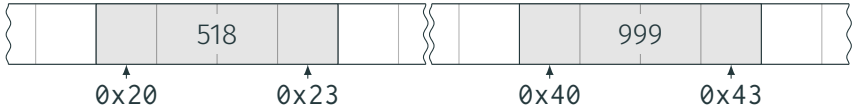

Адресом блока памяти считается номер его самого младшего байта.

int* — указатель на int, т.е. адрес памяти, где лежит int.

Заметьте, у поменяло свое значение, хотя прямых записей в него не было.

0x*** - это шестнадцатеричная запись числа, именно в таком виде принято записывать значения адресов. И именно в таком виде их выводит printf с модификатором %p.

Указатель как адрес




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

```
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
```

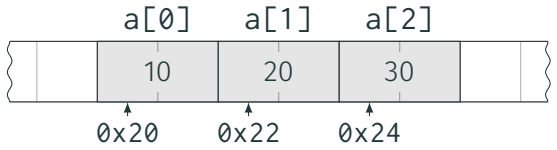

Адресом блока памяти считается номер его самого младшего байта.

int* — указатель на int, т.е. адрес памяти, где лежит int.

Заметьте, у поменяло свое значение, хотя прямых записей в него не было.

0x*** - это шестнадцатеричная запись числа, именно в таком виде принято записывать значения адресов. И именно в таком виде их выводит printf с модификатором %p.

Массивы и указатели



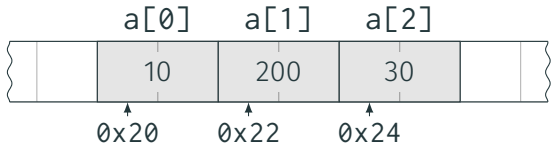

```
short a[3] = {10, 20, 30};

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

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

Массивы и указатели



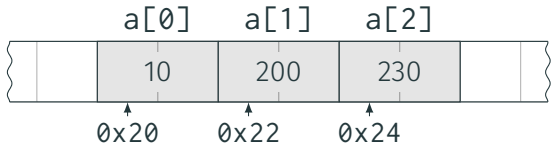

```
short a[3] = {10, 20, 30};          *p1 = 200;

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

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

Массивы и указатели




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

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

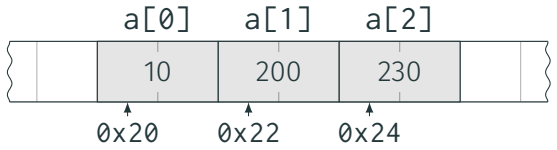
```
*p1 = 200;
*p2 = *p1 + *p2;
```

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

Массивы и указатели




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

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

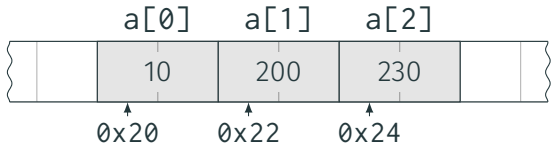
```
*(p+1) = 200;
*(p+2) = *(p+1) + *(p+2);
```

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

Массивы и указатели




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

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

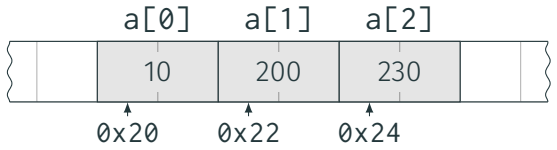
```
p[1] = 200;
p[2] = p[1] + p[2];
// *(x + y) == x[y]
```

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

Массивы и указатели




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

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

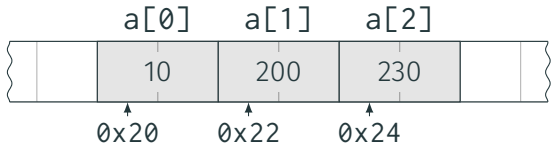
```
p[1] = 200;
p[2] = p[1] + p[2];
// *(x + y) == x[y]
```

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

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]
```

Массив хранится как непрерывный блок памяти, элементы располагаются последовательно.

При прибавлении значения n к указателю, его значение увеличивается на (n * размер типа данных под указателем).

Массивная переменная — всего лишь указатель на начало массива.

Конец второй лекции