Введение в язык программирования C.

Лекция 7
Строки. Ввод-вывод.

Сергей Леонидович Бабичев

Ввод-вывод и строки.

Ввод/вывод и строки

- Процессор оперирует с числами в двоичном представлении.
- Нас обычно интересует текстовая информация, в которой числа представляются в десятичном виде.
- На самом нижнем уровне вывод на экран в терминале читаемой информации возможен только в виде набора символов.
- Системный вывод одного такого символа — трудоёмкая операция.
- Символы перед выводом *буферизуют*, собирая из них массивы и строки.

Преобразование данных при выводе

- Пусть имеется переменная

```
int x = 123;
```

- Что происходит, когда мы выводим её значение?
- Производятся ли какие-либо добавочные операции?
- Производятся.
- Нас не интересуют биты числа `x`.
- Нас интересует *строка*, которая *представляет* это число в десятичном виде.
- Этим и занимается

```
printf("%d", x);
```

- Значение числа `x` преобразуется в строку и затем выводится.

Проектируем функцию преобразования числа в строку

- Возникают вопросы:
  - Что будет входными аргументами функции?
  - Что будет выходными аргументами функции?
  - Как получить итоговую строку?
- Предложим такой прототип функции:

```
void itos(int x, char *str);
```

- Ответами на вопросы будут:
  - Аргумент `x`?
  - Аргумент `str`?
  - Передать в аргументе `str` указатель на массив `char`, в котором поместится результат.

Проектируем функцию преобразования числа в строку

- Как такую функцию использовать?

```
#include <string.h>
#include <stdio.h>
void itos(int x, char *str);

int main() {
    int x = 123;
    char buf[12];
    itos(x, buf);
    puts(buf);
}
```

- Мы *выделили* место для хранения *образа* числа `x` и поместили туда представление этого числа.

Проектируем функцию преобразования числа в строку

- Можно ли было написать

  ```
  char buf[10];?
  ```

- Если бы `x` мы не знали заранее, то нельзя.
- Например, число `-1000000000` занимает в виде строки `1+10+1` символ: без завершающего нулевого байта строка некорректна.
- Поэтому при вызове функции `itos(-1000000000, buf);` мы *залезли* бы в память, которая `buf` не принадлежит.
- Возникла бы страшная ситуация — *переполнение буфера* — бич программистов, пишущих на многих языках.

Пишем функцию преобразования числа в строку

- Как же преобразовать число в строку?
- Строка состоит из *символов*.
- Последнюю цифру строки найти легко: это остаток от деления `x` на 10.
- Но это пока цифра, которая имеет значение, скажем, 3, то есть двоичный код `00000011`.
- Если вывести этот символ в терминал, ничего интересного не увидим, во всяком случае, это не будет тройка.
- При выводе в терминал (или в файл) требуется перекодировать число 3 в код символа `'3'`.
- Значение этого кода зависит от кодировки, принятой на компьютере.
- В кодировке `ASCII` символ `'3'` имеет код 51, а в кодировке `EBCDIC` — код 243.
- Но во всех кодировках символы чисел строго последовательны.
- Так что такой код везде равен `3 + '0'`.

Пишем функцию преобразования числа в строку

- Последний символ известен: это `x%10 + '0'`
- А предпоследний?
- Разделив `x` на 10 мы получим новое число, с которым мы можем провести такую же операцию.
- Закончим всё это, когда число *закончится*, то есть станет равным нулю.
- А куда мы будем помещать код?
- В массив-аргумент `str`.

Первый вариант функции itos

```
void itos(int x, char *str) {
    char *ptr = str;
    while (x != 0) {
        *ptr++ = x%10 + '0';
        x /= 10;
    }
}
```

- Всё правильно?

Первый вариант функции itos

```
void itos(int x, char *str) {
    char *ptr = str;
    while (x != 0) {
        *ptr++ = x%10 + '0';
        x /= 10;
    }
}
```

- Всё правильно?
- Нет, пока много ошибок.
- Например, то, что мы получили — не строка, там нет завершающего нулевого байта.
- Добавим его после окончания цикла.

Второй вариант функции itos

```
void itos(int x, char *str) {
    char *ptr = str;
    while (x != 0) {
        *ptr++ = x%10 + '0';
        x /= 10;
    }
    *ptr = 0;
}
```

- Теперь правильно?

Второй вариант функции itos

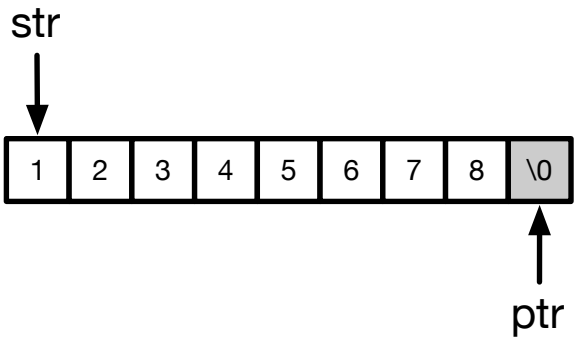
```
void itos(int x, char *str) {
    char *ptr = str;
    while (x != 0) {
        *ptr++ = x%10 + '0';
        x /= 10;
    }
    *ptr = 0;
}
```

- Теперь правильно?
- Почти :) Только вместо 123 получилось 321.
- Нужно развернуть строку.

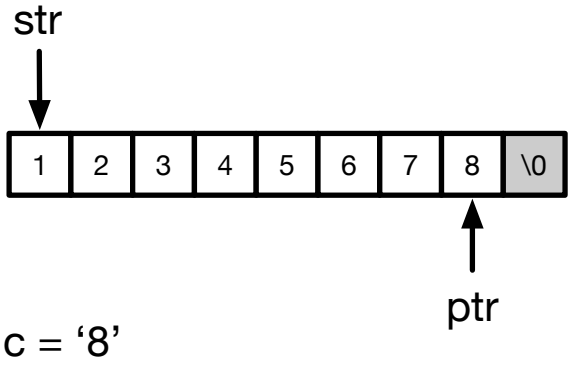
Третий вариант функции itos

```
void itos(int x, char *str) {
    char *ptr = str;
    while (x != 0) {
        *ptr++ = x%10 + '0';
        x /= 10;
    }
    *ptr = 0;
    while (ptr > str) {
        char c = *--ptr;
        *ptr = *str;
        *str++ = c;
    }
}
```

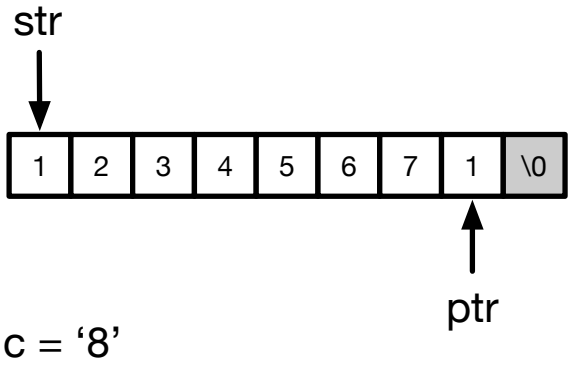
- Строчки, что мы добавили, необычны.
- Но всё объяснимо.



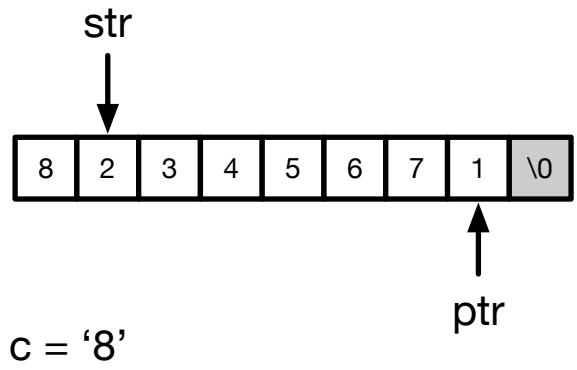
- Начальное состояние. Серым помечен нулевой байт.



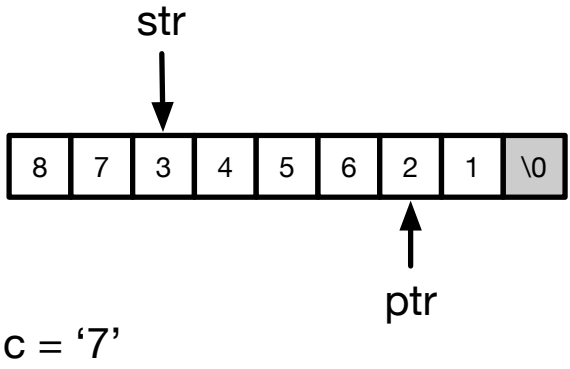
- Первое присваивание `char c = *-ptr;`
- Сначала `ptr` сдвигается влево и по новому адресу извлекается значение `c`, равное '8'.



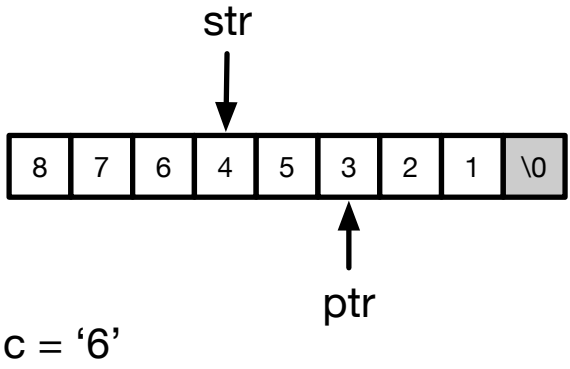
- Второе присваивание `*ptr = *str;`
- По адресу `ptr` присваивается значение по адресу `str`.



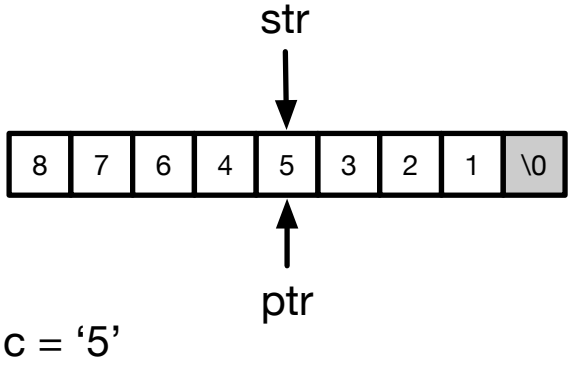
- Третье присваивание `*str++ = c;`
- Сначала по адресу `str` присваивается значение `c`, затем указатель сдвигается вправо.
- Первый и последний элементы строки поменялись местами.
- Указатели тоже сдвинулись по направлению друг к другу и готовы работать дальше.



- После второй итерации цикла ещё пара символов обработана.



- Третья итерация — ещё одна пара символов готова.



- Четвёртая итерация — обработаны все пары. Указатели сравнялись и цикл закончился.
- Обратите внимание: нулевой символ остался на месте!

Пишем функцию преобразования числа в строку

- Готова ли у нас функци itos?
- Не будет ли там переполнения?
- Все ли аргументы она умеет обрабатывать?

Пишем функцию преобразования числа в строку

- Готова ли у нас функци itos?
- Не будет ли там переполнения?
- Все ли аргументы она умеет обрабатывать?
- Переполнения не будет, если мы подадим туда массив длиной не менее 12 символов.
- Она не умеет обрабатывать отрицательные аргументы и нуль.

```
void itos(int x, char *str) {
    if (x == 0) {
        *str++ = '0';
        *str = 0;
        return;
    }
    if (x < 0) {
        *str++ = '-';
        x = -x;
    }
    char *ptr = str;
    while (x != 0) {
        *ptr++ = x%10 + '0';
        x /= 10;
    }
    *ptr = 0;
    while (ptr > str) {
        char c = *--ptr;
        *ptr = *str;
        *str++ = c;
    }
}
```

Пишем функцию преобразования числа в строку

- Разве `printf` не может это сделать самостоятельно?
- Может для восьмеричной, десятичной и шестнадцатеричной систем.
- Если нам нужны другие системы счисления, `printf` бессилен.
- Мы же, заменив 10 на что-то другое, можем написать что-то универсальное.
- Например, вот перевод в систему счисления от двоичной до шестнадцатитичной.
- Не забудьте, что в двоичной системе цифр станет больше и буфер надо расширить.

```
void itos(int x, char *str, int r) {
    if (x == 0) {
        *str++ = '0';
        *str = 0;
        return;
    }
    if (x < 0) {
        *str++ = '-';
        x = -x;
    }
    char *ptr = str;
    while (x != 0) {
        *ptr++ = "0123456789ABCDEF"[x%r];
        x /= r;
    }
    *ptr = 0;
    while (ptr > str) {
        char c = *--ptr;
        *ptr = *str;
        *str++ = c;
    }
}
```