Строки

Строки

```
// это код на C++, ок да?
"string";
```

- `"string"` - это строковый литерал
- `"string"` иммет тип const char[7] (константный массив `char` из 7 элементов)
- В строковом литерале всегда неявно присутствует завершающий символ `\0` (нуль-терминатор).

Строковые литералы

1. Представляют собой массивы символов с `'\0'` на конце

```
std::cout << "ab"[0]  // 'a'
          << "ab"[1]  // 'b'
          << ("ab"[2] == '\0');  // true
```

2. Могут быть скопированы при инициализации (исключение из общего правила)

```
char str[4] = "kek";  // Ok
// str = "lol";          CE (не инициализация)
```

Строковые литералы

3(?). Могут сравниваться (?!)

```
std::cout << ("kek" == "kek" ? "Ну нет..." : "false") << '\n';
std::cout << ("kek" != "lol" ? "Да не может быть..." : "false") << '\n';
```

```
Ну нет...
Да не может быть...
```

Вы же не забыли, что массивы сравниваются не поэлементно?

Сравниваются адреса первых элементов.

Строковые литералы: объяснение

- Массив, с которым связан строковый литерал, лежит в статической (глобальной) области памяти (в таблице строковых литералов).
- Компилятор, анализируя исходный код, помещает каждый попавшийся строковый литерал в отдельный буфер (отдельная строка таблицы).
- Как правило, одинаковые литералы ссылаются на одну и ту же область памяти, поэтому их сравнение путем сравнения указателей может давать верный результат (но это не гарантировано стандартом!).

Строки

- Строка - часть массива элементов `char`, ограниченная символом `\0`
- Правила работы со строками такие же как и с обычными массивами

Строки

- Как создать строку?

```
const char* static_string = "lol";
char stack_string[4] = "kek";
char yet_another_string[] = {'k', 'e', 'k'};
char* heap_string = new char[9];

std::cin >> stack_string; // UB if input is > 3 symbols
std::cin >> heap_string;  // UB if input is > 8 symbols
```

- Где подвох?

Строки

```
const char* static_string = "lol";
char stack_string[4] = "kek";
char yet_another_string[] = {'k', 'e', 'k'}; // не строка! (массив размера 3)
char* heap_string = new char[9];
```

- В строке 3 в отличие от строки 2 нет завершающего нуля!

```
char yet_another_string[] = {'k', 'e', 'k', '\0'}; // теперь строка
```

Строки

Для строк есть удобный интерфейс из большого количества функций для работы с ними, который всем нравится.

`<cstring>`

`std::strcpy` `std::strncpy` `std::strcat` `std::strncat` `std::strxfrm`
`std::strlen` `std::strcmp` `std::strncmp` `std::strcoll` `std::strchr`
`std::strrchr` `std::strspn` `std::strcspn` `std::strpbrk` `std::strstr`
`std::strtok`

Задача

Посчитать количество вхождений символа в строку

```
size_t CountSymbol(const char* str, char symbol) {
    ...
}
```

Задача

Посчитать количество вхождений символа в строку

```
size_t CountSymbol(const char* str, char symbol) {
  size_t counter = 0;
  for (size_t i = 0; i < std::strlen(str); ++i) {
    if (str[i] == symbol) {
      ++counter;
    }
  }
  return counter;
}
```

Задача

Посчитать количество вхождений символа в строку

```
size_t CountSymbol(const char* str, char symbol) {
  size_t counter = 0;
  for (size_t i = 0; i < std::strlen(str); ++i) {  // O(n^2)
    if (str[i] == symbol) {
      ++counter;
    }
  }
  return counter;
}
```

Задача

Посчитать количество вхождений символа в строку

```
size_t CountSymbol(const char* str, char symbol) {
  size_t counter = 0;
  for (; *str != '\0'; ++str) {  // O(n)
    if (*str == symbol) {
      ++counter;
    }
  }
  return counter;
}
```