Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

# Алгоритмы и алгоритмические языки
# Лекция 9

5 октября 2019 г.

# Строки

- Строка — это одномерный массив типа `char`.

  Объявляя массив, предназначенный для хранения строки, необходимо предусмотреть место для символа `'\0'` (конец строки).
- Строковая константа записывается как `"string constant"`.

  В конец строковой константы компилятор добавляет `'\0'`.
- Стандартная библиотека функций работы со строками `<string.h>`, в частности, содержит такие функции, как:

| | |
|---|---|
| `strcpy(s1, s2)` | копирование `s2` в `s1` |
| `strcat(s1, s2)` | конкатенация `s2` и `s1` |
| `strlen(s)` | длина строки `s` |
| `strcmp(s1, s2)` | сравнение `s2` и `s1` в *лексикографическом порядке*: 0, если `s1` и `s2` совпадают, отрицательное значение, если `s1 < s2`, положительное значение, если `s1 > s2` |
| `strchr(s, ch)` | указатель на первое вхождение символа `ch` в `s` |
| `strstr(s1, s2)` | указатель на первое вхождение подстроки `s2` в строку `s1` |

Дома. Прочитайте о функциях `strspn`, `strpbrk`.

Зачем нужна функция `stpcpy`?

# Пример программы работы со строками

```
#include <stdio.h>
#include <string.h>
int main (void) {
  char str1[80], str2[80], smp[3] = "ВМК";

  fgets (str1, 80, stdin); str1[strlen (str1)-1] = '\0';
  fgets (str2, 80, stdin); str2[strlen (str2)-1] = '\0';
  printf ("Строки␣имеют␣длину:␣первая␣%d,␣вторая␣%d\n",
           strlen (str1), strlen (str2));
  if (!strcmp (str1, str2))
    printf ("строки␣равны\n");
  strncat (str1, str2, 80 – strlen (str1) - 1);
  printf ("%s\n", str1);
  sprintf (str1, "Привет,␣%s", smp);
  puts (str1);
  return 0;
}
```

# Пример программы работы со строками

```
#include <stdio.h>
#include <string.h>
int main (void) {
  char str1[80], str2[80], smp[3] = "ВМК";

  fgets (str1, 80, stdin); str1[strlen (str1)-1] = '\0';
  fgets (str2, 80, stdin); str2[strlen (str2)-1] = '\0';
  printf ("Строки␣имеют␣длину:␣первая␣%d,␣вторая␣%d\n",
           strlen (str1), strlen (str2));
  if (!strcmp (str1, str2))
    printf ("строки␣равны\n");
  strncat (str1, str2, 80 – strlen (str1) - 1);
  printf ("%s\n", str1);
  sprintf (str1, "Привет,␣%s", smp);
  puts (str1);
  return 0;
}
```

# Пример программы работы со строками

```
#include <stdio.h>
#include <string.h>
int main (void) {
  char str1[80], str2[80], smp[4] = "ВМК";

  fgets (str1, 80, stdin); str1[strlen (str1)-1] = '\0';
  fgets (str2, 80, stdin); str2[strlen (str2)-1] = '\0';
  printf ("Строки␣имеют␣длину:␣первая␣%d,␣вторая␣%d\n",
           strlen (str1), strlen (str2));
  if (!strcmp (str1, str2))
    printf ("строки␣равны\n");
  strncat (str1, str2, 80 – strlen (str1) - 1);
  printf ("%s\n", str1);
  sprintf (str1, "Привет,␣%s", smp);
  puts (str1);
  return 0;
}
```

# Операция `sizeof`

- Одноместная операция `sizeof` позволяет определить длину операнда в байтах
  - Операнды — типы либо переменные
  - Результат имеет тип `size_t`
- Операция `sizeof` выполняется во время компиляции, её результат представляет собой константу
- `sizeof` помогает улучшить переносимость программ
  - Для определения объема памяти в байтах, нужного для двумерного массива

    `number_of_bytes = d1 * d2 * sizeof (element_type)`

    где `d1` — количество элементов по первому измерению,
    `d2` — количество элементов по второму измерению,
    `element_type` — тип элемента массива
  - Можно поступить и проще:

    `number_of_bytes = sizeof (array_name)`

# Операция `sizeof`

- `sizeof` можно применять только к «полностью» определённым типам. Для массивов это означает:

  размерности массива должны присутствовать в его объявлении,

  тип элементов массива должен быть полностью определён.

- Пример. Если объявление массива имеет вид:
  ```
  extern int arr[];
  ```
  то операция `sizeof (arr)` ошибочна, так как у компилятора нет возможности узнать, сколько элементов содержит массив `arr`.

# Операция sizeof

```
#include <stdio.h>
#include <string.h>
int main (int argc, char **argv) {
  char buffer[10];

  /* копирование 9 символов из argv[1] в buffer;
   sizeof (char) равно 1, число элементов массива
   buffer равно его размеру в байтах */
  strncpy (buffer, argv[1],
           sizeof (buffer) – sizeof (char));
  buffer[sizeof (buffer) - 1] = '\0';
  return 0;
}
```

# Указатели

`&` — операция адресации

`*` — операция разыменования

```
int a = 1;
int *p;
p = &a;
*p = 2;
printf ("Значение␣переменной␣a␣=␣%d\n", *p);
printf ("Адрес␣переменной␣a␣=␣%p\n", p);
```

В результате выполнения фрагмента будет напечатано:

```
Значение переменной a = 2
Адрес переменной a = 0xbffff7a4
```

`&foo` является константой, указатель — переменной

`foo` должен быть l-значением (lvalue)

Печать адреса — модификатор `%p`

Нулевой указатель (никуда не указывающий) — `NULL`
(константа в `stdlib.h`, может не иметь нулевого значения)

# Адресная арифметика

В языке Си допустимы следующие операции над указателями:

- сложение указателя с целым числом;
- вычитание целого числа из указателя;
- вычитание указателей;
- операции отношения и сравнения.

Пример. Пусть `sizeof (int) == 4` и пусть текущее значение `int* p1` равно `2016=0x7E0`.

После операции `p1++` значение `p1` будет `2020=0x7E4` (а не `2017=0x7E1`), после операции `p1-3` — значение `2004=0x7D4`.

> При увеличении (уменьшении) на целое число `i` указатель будет перемещаться на `i` ячеек соответствующего типа в сторону увеличения (уменьшения) их адресов.

# Преобразование типа указателя

Указатель можно преобразовать к другому типу, но такое преобразование типов обязательно должно быть явным.

**Условие**: исходный указатель правильно *выравнен* для целевого типа. Значение указателя сохраняется.

Иногда такое преобразование типов может вызвать непредсказуемое поведение программы.

```
#include <stdio.h>
int main (void) {
  double x = 200.35, y;
  int *p;
  p = (int *)&x; /* &x - double, а p имеет тип int* */
  y = *p;        /* будет ли y присвоено значение 200.35? */
  printf ("значение␣x␣равно␣%f\n", x);
  printf ("значение␣y␣равно␣%f\n", y);
  return 0;
}
```

# Преобразование типа указателя

Типичный вывод (GCC, Linux):

```
значение x равно 200.350000
значение y равно 858993459.000000
```

> В присваивании `y = *p;` загрузка `*p` считывает только первые **четыре** байта области памяти с адресом `&x` (т.к. `sizeof (int)` в данном случае равен 4)
>
> В представлении 200.35 в формате числа `double` первые четыре байта соответствуют целому числу 858993459

Таким образом, необходимо учитывать, что операции с указателями выполняются **в соответствии с базовым типом указателя**.