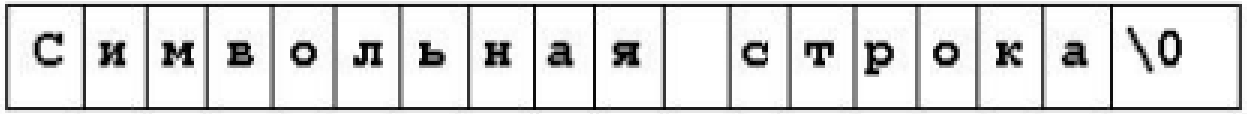
Строки

Работа со строками в стиле C

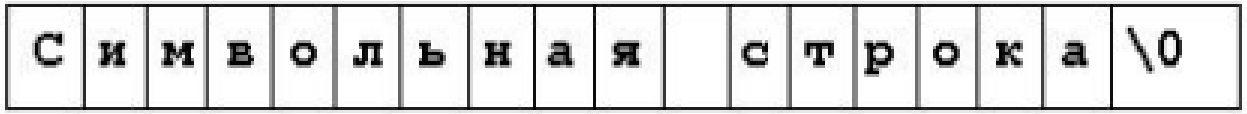
- Строковая константа — это последовательность из нуля или более символов, заключенных в кавычки.
- Терминирующий ноль (\0) – символ завершения строки, он не выводится на печать и в таблице кодов ASCII имеет номер 0.




```
char line[5] = {'C','a','t','!','\0’ };
void main () {
      cout << "Word: ";
      for (int i=0; i < 5; ++i)
            cout << line[i];
}
```

Немного не удобно, но можно иначе

- Строковая константа — это последовательность из нуля или более символов, заключенных в кавычки.
- Терминирующий ноль (\0) – символ завершения строки, он не выводится на печать и в таблице кодов ASCII имеет номер 0.




```
char line[] = "Cat!"; // Инициализация строкового массива.
void main () {
      cout << "Word: ";
      for (int i=0; i < 5; ++i)
            cout << line[i];
}
```

- В описании массива char line[] не указан размер – компилятор сам подсчитает количество символов.

```
/* Название k-го месяца */
char *month_name (int k) {
      static char *name[] = {
                  "none","January","February","March",
                  "April","May","June","July","August",
                  "September","October","November","December"
      };
      return (k < 0 || k > 12)? name[0]:name[k];
}

void main () {
      for (int i=1; i <= 12; i++)
            cout << "Month " << i << " - " << month_name(i) <<"\n";
}
```

Некоторые функции работы со строками:

- `int atoi(const char* s)` — Преобразует строку s в число типа int. Возвращает значение или нуль, если строку преобразовать нельзя. `stdlib.h`
- `char* itoa(int value, char* s, int radix)` — Преобразует значение целого типа `value` в строку `s`. Возвращает указатель на результирующую строку. Значение `radix` — основание системы счисления, используемое при преобразовании (от 2 до 36). `stdlib.h`.
- `int getchar()` — Возвращает значение символа (если он есть), который пользователь набрал на клавиатуре. После ввода символа нужно нажать клавишу Enter. `stdio.h`.
- `int strlen(const char* s)` — Возвращает длину строки `s` — количество символов, предшествующих терминирующему нулю.

- Определение длины строки:

```
void main () {
      char* str = "Any old string…";
      int len = strlen(str);
}

const int MAXLEN = 256;
int main () {
      char string[MAXLEN];
      cout << "Input string: ";
      gets(string);
      cout << endl;
      cout << "String: " << string << endl;
      cout << "Length = " << strlen(string) << endl;
}
```

- Копирование строк:

```
char* str1 = "Hello";
char* str2 = "World";
str1 = str2;
```

Будет ли это работать?

- Копирование строк:

```
char* str1 = "Hello";
char* str2 = "World";
str1 = str2;
```

Будет ли это работать?

```
char* str1 = new char[10];
char* str2 = new char[10];
str1 = str2; // мы потеряли адрес Hello
```

- Вместо копирования символов будет скопирован указатель `c2` в `c1`. Адрес в `c1` перезапишется, потенциально потеряв информацию, адресуемую указателем.

- Копирование строк:

```
char* str1 = "Hello";
char* str2 = "World";
str1 = str2;
```

Будет ли это работать?

```
char* str1 = new char[10];
char* str2 = new char[10];
str1 = str2; // мы потеряли адрес Hello
```

- Вместо копирования символов будет скопирован указатель `c2` в `c1`. Адрес в `c1` перезапишется, потенциально потеряв информацию, адресуемую указателем.

```
strcpy(str1, str2);
```

- Ответственность за то, что принимающая `str1` имеет достаточно места для хранения копии `str2` исключительно на программисте.

- Конкатенация строк:

```
char original[128] = "Hello";
strcat(original, " everybody!");

Output: Hello everybody!
```

- Функция `strcat(char*, char*)` возвращает адрес результирующей строки (совпадающий с ее первым параметром) и может использоваться как каскад нескольких вызовов функций:

```
strcat(strcat(c1, c2), c3);
```

- Ответственность за то, что принимающая `c1` имеет достаточно места для добавления к ней `c2` исключительно на программисте.

- Однако соединить 3 строки в одну не так тривиально как кажется:

```
char* surname = new char[128];
char* name = new char[128];
char* pname = new char[128];

// … fill surname, name and pname

char *result = new char[    strlen(surname) +
                           strlen(name) +
                           strlen(pname) + 3];

strcat(strcat(strcpy(result, surname), " "),name);
strcat(strcat(result, " "), pname);

delete [] surname; delete [] name; delete [] pname;
```

- А теперь поищем символы в строке:

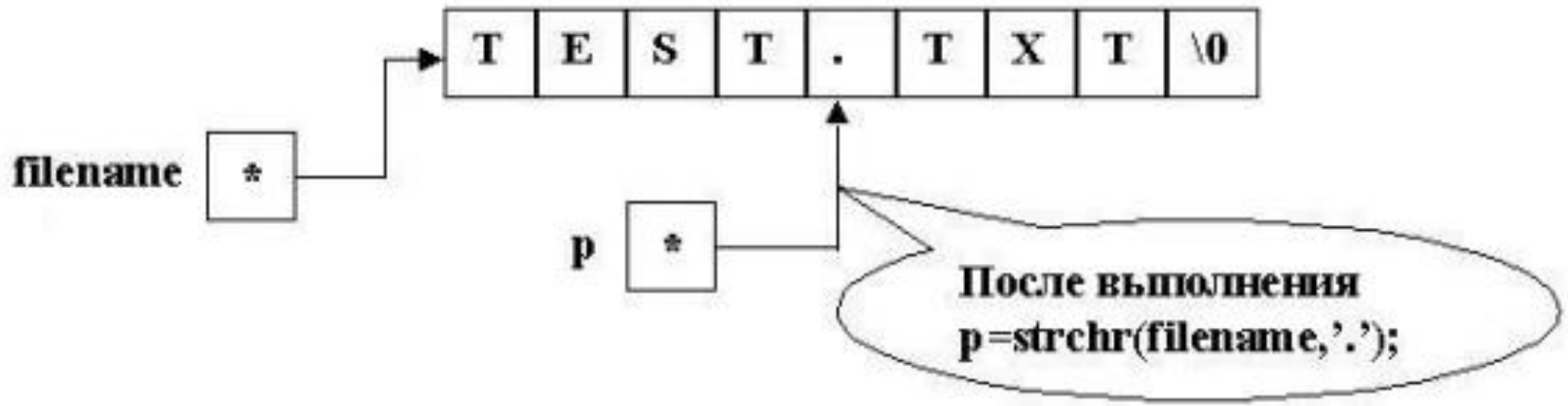
```
char filename[128] = new char[128];
gets(filename);

if(strchr(filename, '.')) {
      cout << "File has extension!" << endl;
}
else {
      strcat(filename, ".TXT");
}

cout << "File name is " << filename << endl;
delete [] filename;
```

- `strchr(char* s1, char c)` возвращает указатель на символ `c` в строке `s1`. В противном случае возвращает 0.

```
char* p = strchr(filename, '.');
```



- Некорректно вызывать `delete [] p`, поскольку p лишь указатель на начало подстроки в строке `filename`.
- Корректный `delete [] filename`.

- Аналогичный поиск подстроки:

```
char filename[128] = new char[128];
gets(filename);
strupr(filename); // возвести строку в верхний регистр
char * p = strstr(filename, ".TXT");
if(p) {
      cout << "File has extension!" << endl;
}
else {
      p = strchr(filename,'.’);
      if(p) *p = NULL;        // удалить любое другое расширение
      strcat(filename, ".TXT");
}

cout << "File name is " << filename << endl;
delete [] filename;
```