1. Одномерные и двумерные символьные массивы.

1.1. Строки и массивы строк

Строка - это одномерный массив символов, который заканчивается нулевым символом конца строки. В качестве нулевого символа выступает символ `'\0'`. Таким образом, строка содержит символы, которые ее составляют, и нулевой символ. При объявлении массива символов, который предназначен для хранения строки, необходимо предусмотреть место для нулевого символа, то есть при его объявлении указать размер этого массива на единицу больше, чем число предполагаемых символов в массиве. Например, объявление массива из 10 символов должно выглядеть следующим образом:

```
#define LEN 10
char name[LEN + 1];
```

При этом величина `LEN + 1` влияет всего лишь на количество выделяемой для него памяти, сам массив может быть меньше этой величины. Таким образом, размер массива при его объявлении – это только максимально возможная длина строки, но, ни в коем случае, не текущий ее размер. Таким образом, в отличие от числовых массивов, длина строкового массива может как бы «изменяться» в пределах его размера.

Рисунок 9.1. Иллюстрация разницы между размером символьного массива и длины строки

При объявлении строки она также может быть сразу инициализирована:

```
#define LEN_1 20
#define LEN_2 29
char str1[LEN_1 + 1] = "строка\n";
char str2[LEN_2 + 1] = {"Test string"};
```

В первом случае выделяется память для 20 символов, а заполняются только семь из них (’\n’ – это один символ). Во втором случае заполняются одиннадцать символов из 29ти выделенных в памяти.
Существует возможность вообще не указывать размер строки при ее объявлении. В этом случае строку нужно инициализировать в обязательном порядке. Размер такой переменной-строки вычисляется, исходя из длины инициализирующей строки.

```
char special_str[] = "Это массив символов";
```

Для данного случая выделится память для хранения 20 символов.

Функции ввода/вывода строк:

- `gets();`
- `puts();`
- `fgets();`
- `fputs();`
- `scanf();`
- `printf();`
- `fscanf();`
- `fprintf();`
- `sscanf();`
- `sprintf().`

Отметим несколько особенностей функций ввода/вывода строк. Подробнее эти функции рассмотрены в Разделе 11.1.

Функция	**Особенности функции**
`scanf();`	Считывание символов до первого пробела
`fscanf();`	
`gets();`	`'\n'` не дописывается в строку
`fgets();`	`'\n'` дописывается в строку
`puts();`	`'\n'` добавляется после вывода
`fputs();`	`'\n'` не добавляется после вывода

Массив строк – это список из нескольких строк. Например, текстовый файл можно рассматривать как набор отдельных строк, каждую из которых можно записать в массив строк, который будет отображать содержимое текстового файла.

Рисунок 9.2. Массив строк с одинаковой максимальной длиной

Рисунок 9.3. Массив строк с различной максимальной длиной каждой строки

Объявление массивов строк происходит аналогично двумерным числовым массивам. Индекс слева определяет количество строк, а правый – максимальное количество символов в строке. Объявим массив из 25ти строк с максимальной длиной каждой строки 48 символов:

```
#define LEN_ARR 25
#define LEN_STR 48
char str_array[LEN_ARR][LEN_STR];
```

Так как слово – это тоже строка, то прочитав текстовый файл, можно образовать трехмерный символьный массив таким образом, что первое измерение будет отвечать за номер строки в файле, второе – за номер слова в строке, а третье – за номер символа в слове.

```
#define NUM_STR 15 //количество строк в файле
#define NUM_WRD 10 //количество слов в строке
#define LEN_WRD 100 // длина слова

char words_array[NUM_STR][NUM_WRD][LEN_WRD];
```

В этом примере используется допущение, что в каждой строке одинаковое количество слов. Этого можно избежать при применении динамической памяти.

```
#define NUM_STR 10
int num_wrds[NUM_STR] = {3, 5, 1, 8, 9, 11, 3, 3, 2, 7};
#define LEN_WRD 120

char ** word_list;
word_list = (char**)malloc(NUM_STR * sizeof(char*));
int i = 0;
for(i = 0; i < NUM_STR; i++) {
  word_list[i] =
```

```
                (char*)malloc(num_wrds[i]* sizeof(char*));
    }
    int j = 0;
    for(i = 0; i < NUM_STR; i++) {
      for(j = 0; j < num_wrds[i]; j++) {
        word_list[i][j] =
                (char*)malloc(LEN_WRD * sizeof(char[LEN_WRD]));
      }
    }
```

Здесь массив `num_wrds` соответствует количеству слов в строке `i + 1`, для примера массив задан явно при объявлении.

1.2. Функции стандартной библиотеки string.h

Для упрощения работы со строками в языке Си определено множество различных функций, которые содержатся в заголовочном файле библиотеки `string.h` ниже представлены несколько из них.

Функция	**Описание**
`strlen(str)`	Определение длины строки `str`
`strcmp(str_1, str_2)`	Сравнение строки `str_1` и строки `str_2`
`strcat(str_1, str_2)`	Дописать строку `str_2` в строку `str_1`
`strcpy(str_1, str_2)`	Копирование строки `str_2` в строку `str_1`
`strchr(str, symbol)`	Возвращает первую позицию символа symbol в строке str (поиск с начала строки)
`strrchr(str, symbol)`	Возвращает первую позицию символа `symbol` в строке `str` (поиск с конца строки)
`strpbrk(str_1, str_2)`	Возвращает первую позицию любого из символов строки `str_2` в строке `str_1` (поиск с начала строки)
`strstr(str_1, str_2)`	Возвращает первую позицию вхождения строки `str_2` в строку `str_1` (поиск с начала строки)

1.3. Задачи с использованием строк и массивов строк

Пример. Вывести строку на экран, вставляя пробелы между ее буквами.

```
#define LEN 250
char sample_string[LEN];
gets(sample_string);
int i = 0;
```

```
for(i = 0; i < strlen(sample_string); i++) {
  printf("%c ", sample_string[i]);
}
```

Пример. Определить совпадают ли два введенных с клавиатуры адреса.

```
#define LEN 25
char address_Ann[LEN], address_Mary[LEN];
gets(address_Ann);
gets(address_Mary);
if (!strcmp(address_Ann, address_Mary))
    printf("Адреса совпадают");
else
    printf("Это разные адреса");
```

Так как функция **strcmp()** возвращает ноль, если строки одинаковы, то для выполнения условия ставим отрицание. Таким образом, если строки равны, то значение функции **strcmp()** ноль будет инвертирован в единицу и условие выполнится.

Пример. Найти длину строки без использования стандартной функции

```
#define LEN 250
char sample_string[LEN];
gets(sample_string);
int i = 0, length = 0;
while(sample_string[i] != '\0') {
  length++;
  i++;
}
printf("%d", length);
```

Пример. Разделить введенную с клавиатуры строку на две строки.

```
#define LEN 250
char sample_string[LEN], new_str_1[LEN], new_str_2[LEN];
gets(sample_string);
int i = 0, k = 0;

for(i = 0, k = 0; i <strlen(sample_string) / 2; i++, k++) {
  new_str_1[i] = sample_string[i];
}
new_str_1[k] = '\0';
```

```
for(i = strlen(sample_string) / 2, k = 0;
                          i < strlen(sample_string); i++, k++) {
 new_str_1[k] = sample_string[i];
}
new_str_1[k] = '\0';
```

Пример. Определить, какое место в стоке является первым вхождением в строку заданного символа.

```
#define LEN 50
char string_1[LEN + 1], letter = 0;
gets(string_1);
letter = getchar();
char * position = strchr(string_1, letter);
if(position == 0) {
  printf("Такого символа нет в строке\n");
}
else {
  printf("Символ найден в строке\n");
  int i = 0, ind = 0;
  for(i = 0; i < strlen(string_1); i++) {
    if ((string_1 + i) == position) {
      ind = i;
      break;
    }
  }
  printf("Указатель буквы %c = %p\n", letter, position);
  printf("Позиция символа %c в строке = %d", letter, ind + 1);
}
```

Пример. Удалить из строки заданную подстроку.

```
#define LEN 250
char smpl_string[LEN], str_to_delit[LEN];
gets(smpl_string);
gets(str_to_delit);

char * position = strstr(smpl_string, str_to_delit);
int i = 0, ind = 0;
for(i = 0; i < strlen(smpl_string); i++) {
  if ((smpl_string + i) == position) {
    ind = i;
```

```
      break;
    }
  }

  int k = 0;
  for(i = ind, k = ind + strlen(str_to_delit);
                          k < strlen(smpl_string); i++, k++) {
    smpl_string[i] = smpl_string[k];
  }
  smpl_string[strlen(smpl_string) - strlen(str_to_delit)] = '\0';
  puts(sample_string);
```

Пример. Вводить строки с клавиатуры и записывать их в массив строк `str_arr[10][50]` только тогда, когда длина введенной строки превышает `15` символов.

```
#define LEN_ARR 10
#define LEN_STR 50
char str_arr[LEN_ARR][LEN_STR];
for(i = 0; i < LEN_ARR;  ) {
  gets(str_arr[i]);
  if (strlen(str_arr[i]) > 15) {
    i++;
  }
}
```

Пример. В массиве из `50` строк, найти и вывести на экран самые короткие из них.

```
#define LEN_ARR 50
#define LEN_STR 200
char str_arr[LEN_ARR][LEN_STR];

int i = 0;
for(i = 0; i < LEN_ARR; i++) {
  gets(str_arr[i]);
}

//начальное значение минимальной длины - длина первой строки
int min_len = strlen(str_arr[0]);

// находим минимальную длину строки
for(i  = 0; i < LEN_ARR; i++) {
  if (strlen(str_arr[i]) > min_len) {
    min_len = strlen(str_arr[i]);
```

```
    }
  }
  //выводим на экран все строки, длина которых равна минимальной
  for(i = 0; i < LEN_ARR; i++) {
    if (strlen(str_arr[i]) == min_len) {
      puts(str_arr[i]);
    }
  }
```

Пример. Отсортировать массив строк по возрастанию длины строк.

```
  #define LEN 250
  #define LEN_ARR 25
  char string_array[LEN_ARR][LEN], temp_str[LEN];
  int i = 0;
  for(i = 0; i < LEN_ARR; i++) {
   gets(string_array[i]);
  }

  // численный массив с длинами каждой строки
  int  str_lenght[LEN_ARR];
  for(i = 0; i < LEN_ARR; i++) {
    str_lenght[i] = strlen(string_array[i]);
  }

  // сортировка массива строк одновременно с сострировкой массива длин строк
  int k = 1, tmp = 0;
  while(k > 0) {
    k = 0;
    for(i = 0; i < LEN_ARR - 1; i++) {
      if(str_lenght[i] > str_lenght[i + 1]) {
        k++;
        tmp = str_lenght[i];
        str_lenght[i] = str_lenght[i + 1];
        str_lenght[i + 1] = tmp;

        strcpy(temp_str, string_array[i]);
        strcpy(string_array[i], string_array[i + 1]);
        strcpy(string_array[i + 1], temp_str);
      }
    }
  }
```

1.4.Работа с текстовыми файлами

Работа с текстовыми файлами всегда начинается с чтения исходного текстового файла.

Чтение текстового файла

Чтение до определенного момента	**Чтение до конца файла**
Чтение `N` символов	Чтение всех символов
Чтение `N` слов	Чтение всех слов
Чтение `N` строк	Чтение всех строк

Пример. Прочитать из файла `text_file.txt` и вывести на экран `7` символов, затем `15` слов и еще `9` строк.

```
#define LEN_STR 250
#define LEN_WRD 250
FILE *file;
char symbol = 0, str_wrd[LEN_WRD], str_array[LEN_STR];

file = fopen("text_file.txt", "r");
int i = 0;

// чтение файла по символам
for(i = 0; i < 7; i++) {
 symbol = fgetc(file);
}

// чтение файла по словам
for(i = 0; i < 15; i++) {
 fscanf(file, "%s", str_wrd);
}

// чтение файла по строкам
for(i = 0; i < 9; i++) {
 fgets(str_array, LEN_STR, file);
}

fclose(file);
```

Символ конца файла EOF

При достижении конца файла функция **fgetc()** возвращает символ EOF. Этот символ означает конец файла (End Of File). В численном значении символ EOF равен минус единице. Поскольку у каждого символа есть свой код, и этот код является положительным числом, то минус единица не может быть кодом какого-либо символа.

Использование символа EOF может быть удобно только при посимвольном чтении файла. Если файл читается с помощью **fgets()** или **fscanf()**, то для определения достижения конца файла удобнее использовать функцию **feof()**. Эта функция универсальна и может использоваться при любом способе чтения файла (в том числе при чтении по одному символу).

Главное отличие **feof()** и EOF состоит в том, что при использовании EOF программа проверяет равенство какой-либо из символьных переменных значению конца файла. При использовании функции **feof()** программа проверяет, не является ли последний прочитанный символ равным значению конца файла.

Пример. Вывести на экран все содержимое файла по одному символу.

```
char letter = 0;
// пока символ не равен символу конца файла
while ((letter = fgetc(file)) != EOF) {
    putchar(letter);
}
fclose(file);
```

Пример. Вывести на экран все строки из файла, пропуская между ними пустую строку.

```
#define LEN_STR 200
char str[LEN_STR];
// пока последний прочитанный символ не равен концу файла
while (!feof(file)) {
  fgets(str, LEN_STR, file);
  puts(str);
}
fclose(file);
```

Здесь функция **fgets()** считывает из файла строку вместе с символом '\n' и при записи в строковую переменную добавляет символ конца строки '\0'. Таким образом, при выводе такой строки курсор будет в любом случае переходить на новую строку. Если же выводить ее на экран с помощью функции **puts()**, то переход на новую строку будет двойным.

Все действия с файлами происходят с помощью:

- действия с использованием только исходного файла;
- используется массив строк;
- используется вспомогательный файл.

Использование только исходного файла

Пример. Прочитав файл, определить, сколько слов в этом файле начинается со строчной или прописной буквы «a».

```
#define LEN_STR 50
char word[LEN_STR];
int kol_slov = 0;
while (!feof(file)) {
  fscanf(file, "%s", word); //считываем слово
  if ((word[0] == 'a') || (word[0] == 'A'))
    kol_slov++;
}
printf("Количество слов, начинающихся
                          с буквы\'a\' равно %d\n", kol_slov);
fclose(file);
```

Пример. Прочитав файл, определить, сколько в этом файле слов длиннее 5ти символов и вывести на экран все остальные слова.

```
#define LEN_STR 50
char word[LEN_STR];
int kol_slov = 0;
while (!feof(file)) {
  fscanf(file, "%s", word);
  if (strlen(word) > 5)
    kol_slov++;
  else
    printf("%s ", word);
}
printf("Кол-во слов = %d\n\n", kol_slov);
fclose(file);
```

Пример. Прочитать из файла 7 строк и записать их в массив строк. Вывести массив строк на экран.

```
#define LEN_ARR 7
#define LEN_STR 50
char str_arr[LEN_ARR][LEN_STR], last_symbol = 0;
int i = 0, len = 0;
FILE * file;
```

```
file = fopen("input.txt", "r");
for(i = 0; i < LEN_ARR; i++) {
  //чтение строки
  fgets(str_arr[i], LEN_STR, file);

  //находим чему равен последний символ в строке
  len = strlen(str_arr[i]);
  last_symbol = str_arr[i][len - 1];

  // если последний символ = '\n' меняем его на '\0'
  if (last_symbol == '\n')
    str_arr[i][len - 1] = '\0';
}
fclose(file);

//вывод массива строк на экран
for(i = 0; i < LEN_ARR; i++) {
  puts(str_arr[i]);
}
```

При чтении файла функцией `fgets()` в конце каждой строки (кроме последней строки) записывается символ перехода на новую строку. Передвигая символ конца строки на место символа переноса строки, мы избавляемся от этого нежелательного явления.

Использование массива строк

Пример. Прочитав исходный текстовый файл, отсортировать его строки по убыванию длины.

```
#define LEN_STR 250
FILE * file;
char temp_str[LEN_STR];
int num_str = 0;
//подсчет количества строк в файле
file = fopen("input.txt", "r");
while(!feof(file)) {
  fgets(temp_str, LEN_STR, file);
  num_str++;
}
fclose(file);

//выделение динамической памяти для массива строк
char ** str_array;
str_array = (char**)malloc(num_str * sizeof(char*));
```

```
int i = 0;
for(i = 0; i < num_str; i++) {
  str_array[i] = (char*)malloc(LEN_STR * sizeof(char));
}

//заполнение массива строк, символ "\n" удаляем из конца строки
file = fopen("input.txt", "r");
char last_symbol = 0;
for(i = 0; i < num_str; i++) {
  fgets(str_array[i], LEN_STR, file);
  last_symbol = str_array[i][strlen(str_array[i]) - 1];
  if(last_symbol == '\n') {
    str_array[i][strlen(str_array[i]) - 1] = '\0';
  }
}
fclose(file);

// численный массив с длинами каждой строки
int * str_lenght;
str_lenght = (int*)malloc(num_str * sizeof(int));

for(i = 0; i < num_str; i++) {
  str_lenght[i] = strlen(str_array[i]);
}

//сортировка двух массивов методом пузырька
int k = 1, tmp = 0;
while(k > 0) {
  k = 0;
  for(i = 0; i < num_str - 1; i++) {
    if(str_lenght[i] < str_lenght[i + 1]) {
      k++;
      tmp = str_lenght[i];
      str_lenght[i] = str_lenght[i + 1];
      str_lenght[i + 1] = tmp;

      strcpy(temp_str, str_array[i]);
      strcpy(str_array[i], str_array[i + 1]);
      strcpy(str_array[i + 1], temp_str);
    }
  }
}
```

```
free(str_lenght);

//вывод отсортированного массива строк в исходный файл
file = fopen("input.txt", "w");
for(i = 0; i < num_str; i++) {
  fprintf(file, "%s\n", str_array[i]);
}
fclose(file);

for(i = 0; i <num_str; i++) {
  free(str_array[i]);
}
free(str_array);
```

Пример. Прочитав исходный текстовый файл, образовать новый текстовый файл из неповторяющихся слов первого файла.

```
#define LEN_STR 250
char temp_wrd[LEN_STR];
int num_wrd = 0;

FILE * file;
file = fopen("input.txt", "r");
//считаем количество слов в исходном файле
while(!feof(file)) {
  fscanf(file, "%s", temp_wrd);
  num_wrd++;
}
fclose(file);

//выделяем динамическую память для хранения массива слов
char ** wrd_array;
wrd_array = (char**)malloc(num_wrd * sizeof(char*));

int i = 0;
for(i = 0; i < num_wrd; i++) {
  wrd_array[i] = (char*)malloc(LEN_STR * sizeof(char));
}

//заполняем массив слов
file = fopen("input.txt", "r");
for(i = 0; i < num_wrd; i++) {
```

```
      fscanf(file, "%s", wrd_array[i]);
    }
    fclose(file);

    //открываем итоговый файл для записи
    file = fopen("output", "w");
    int k = 0, count = 0;

    for (k = 0; k < num_wrd; k++) {

    //считаем сколько раз слово встечается в массиве слов
      count = 0;
      for(i = 0; i < num_wrd; i++) {
        if(!strcmp(wrd_array[i], wrd_array[k])) {
          count++;
        }
      }
    //если слово встретилось только один раз, значит, оно не
повторяется
      if (count == 1) {
        fprintf(file, "%s ", wrd_array[k]);
      }
    }
    fclose(file);

    for(i = 0; i <num_wrd; i++) {
      free(wrd_array[i]);
    }
    free(wrd_array);
```

Использование промежуточного файла

Пример. Определить сколько слов в каждой строке.

```
    #define LEN_STR 250
    FILE *file, *tmp_file;
    char str_array[LEN_STR], tmp_wrd[LEN_STR];
    int count = 0;
    //открываем исходный файл для чтения
    file = fopen("input.txt", "r");
    int num_str = 0;
    //читаем одну строку из исходного файла и записываем ее во
временный файл
    while(!feof(file)) {
```

```
    fgets(str_array, LEN_STR, file);
    num_str++;

//запись строки во врменный файл
    tmp_file = fopen("tmp.txt", "w");
    fputs(str_array, tmp_file);
    fclose(tmp_file);

//читаем промежуточный файл по словам и считаем их количество
    tmp_file = fopen("tmp.txt", "r");
    count = 0;
    while(!feof(tmp_file)) {
      fscanf(tmp_file, "%s", tmp_wrd);
      count++;
    }
    fclose(tmp_file);

    printf("В %dй строке %d слов\n", num_str, count);
}
fclose(file);
```

Пример. На основе исходного файла создать новый файл таким образом, чтобы в каждой строке слова располагались в обратном порядке.

```
#define LEN_STR 250
FILE *file, *tmp_file, *out_file;
char str_array[LEN_STR], tmp_wrd[LEN_STR];
int count = 0;
//открываем исходный файл для чтения, итоговый файл для записи
file = fopen("input.txt", "r");
out_file = fopen("outtput.txt", "w");

char last_symbol = 0;
//читаем исходный файл до конца файла
while(!feof(file)) {
//читаем строку и удаляем в ее конце символ "\n"
    fgets(str_array, LEN_STR, file);
    last_symbol = str_array[strlen(str_array) - 1];
    if(last_symbol == '\n') {
      str_array[strlen(str_array) - 1] = '\0';
    }
    //записываем строку в промежуточный файл
    tmp_file = fopen("tmp.txt", "w");
```

```
  fputs(str_array, tmp_file);
  fclose(tmp_file);

//читаем промежуточный файл по словам и считаем их количество
  tmp_file = fopen("tmp.txt", "r");
  count = 0;
  while(!feof(tmp_file)) {
    fscanf(tmp_file, "%s", tmp_wrd);
    count++;
  }
  fclose(tmp_file);

//объявляем динамический массив строк для записи слов
  char ** wrd_array;
  wrd_array = (char**)malloc(count * sizeof(char*));

  int i = 0;
  for(i = 0; i < count; i++) {
    wrd_array[i] = (char*)malloc(LEN_STR * sizeof(char));
  }

//еще раз читаем файл и заполняем массив строк словами
  tmp_file = fopen("tmp.txt", "r");
  for(i = 0; i < count; i++) {
    fscanf(tmp_file, "%s", wrd_array[i]);
  }
  fclose(tmp_file);

//делаем обратную перестановку в массиве строк
  for(i = 0; i < count / 2; i++) {
    strcpy(tmp_wrd, wrd_array[i]);
    strcpy(wrd_array[i], wrd_array[count - 1 - i]);
    strcpy(wrd_array[count - 1 - i], tmp_wrd);
  }
//выводим инвертированный массив строк в итоговый файл
  for(i = 0; i < count; i++) {
    fprintf(out_file, "%s ", wrd_array[i]);
  }
  fprintf(out_file, "\n");

  for(i = 0; i <count; i++) {
    free(wrd_array[i]);
  }
```

```
      free(wrd_array);
    }

    fclose(out_file);
    fclose(file);
```

Пример. Определить в какой из самых длинных строк меньше всего слов.

```
    #define LEN_STR 250
    FILE *file, *tmp_file;
    char str_array[LEN_STR], tmp_wrd[LEN_STR];
    char last_symbol = 0;
    int max_length = 0;

    //определение максимальной длины строки в файле
    file = fopen("input.txt", "r");
    while(!feof(file)) {
      fgets(str_array, LEN_STR, file);
      last_symbol = str_array[strlen(str_array) - 1];
      if(last_symbol == '\n') {
        str_array[strlen(str_array) - 1] = '\0';
      }
      if(strlen(str_array) > max_length) {
       max_length = strlen(str_array);
      }
    }
    fclose(file);

    //среди строк с максимальной длиной находим минимальное
количество слов
    file = fopen("input.txt", "r");
    int min_wrd = 1000, count = 0;;
    while(!feof(file)) {
      fgets(str_array, LEN_STR, file);
      last_symbol = str_array[strlen(str_array) - 1];
      if(last_symbol == '\n') {
        str_array[strlen(str_array) - 1] = '\0';
      }
    //строки с максимальной длиной записываются во временный файл
      if(strlen(str_array) == max_length) {
        tmp_file = fopen("tmp.txt", "w");
        fprintf(tmp_file, "%s", str_array);
        fclose(tmp_file);
```

```
    //поиск количества слов во временном файле
        count = 0;
        tmp_file = fopen("tmp.txt", "r");
        while(!feof(tmp_file)) {
          fscanf(tmp_file, "%s", tmp_wrd);
          count++;
        }
        fclose(tmp_file);
    //сравнение текущего количества слов с минимальным
        if(count < min_wrd) {
          min_wrd = count;
        }
      }
    }
    fclose(file);

    //еще раз проходим по исходному файлу, находим строки с
максимальной длиной и выбираем из них строки с минимальным
количеством слов
    file = fopen("input.txt", "r");
    while(!feof(file)) {
      fgets(str_array, LEN_STR, file);
      last_symbol = str_array[strlen(str_array) - 1];
      if(last_symbol == '\n') {
        str_array[strlen(str_array) - 1] = '\0';
      }
      if(strlen(str_array) == max_length) {
        tmp_file = fopen("tmp.txt", "w");
        fprintf(tmp_file, "%s", str_array);
        fclose(tmp_file);

        int count = 0;
        tmp_file = fopen("tmp.txt", "r");
        while(!feof(tmp_file)) {
          fscanf(tmp_file, "%s", tmp_wrd);
          count++;
        }
        fclose(tmp_file);

        if(count == min_wrd) {
          printf("Искомая строка - %s содержит %d слов\n",
                                              str_array, min_wrd);
```

```
    }
  }
}
fclose(file);
```