Строки и файлы

В C++ есть два вида строк: С-строки и класс стандартной библиотеки C++ `string`. С-строка представляет собой массив символов, завершающийся символом с кодом 0[1]. Класс `string` более безопасен в использовании, чем С-строки, но и более ресурсоемок. Для грамотного использования этого класса требуется знание объектно-ориентированного программирования, поэтому мы рассмотрим его во второй части практикума, а на этом семинаре ограничимся рассмотрением С-строк.

Описание строк

Память под строки, как и под другие массивы, может выделяться как компилятором, так и непосредственно в программе. Длина динамической строки может задаваться выражением, длина не-динамической строки должна быть только константным выражением. Чаще всего длина строки задается частным случаем константного выражения — константой. Удобно задавать длину с помощью именованной константы, поскольку такой вариант, во-первых, лучше читается, а во-вторых, при возможном изменении длины строки потребуется изменить программу только в одном месте:

```
const int len_str = 80;
char str[len_str];
```

При задании длины необходимо учитывать завершающий нуль-символ. Например, в строке, приведенной выше, можно хранить не 80 символов, а только 79. Строки можно при описании инициализировать строковыми константами, при этом нуль-символ в позиции, следующей за последним заданным символом, формируется автоматически:

```
char a[100] = "Never trouble trouble";
```

[1] Этот вид строк, как вы догадались, пришел в C++ из языка C.

Если строка при определении инициализируется, ее размерность можно опускать (компилятор сам выделит память, достаточную для размещения всех символов строки и завершающего нуля):

```
char a[] = "Never trouble trouble";          // 22 символа
```

Для размещения строки в динамической памяти надо описать указатель на char, а затем выделить память с помощью new или malloc (первый способ предпочтительнее):

```
char *p = new char [m];
char *q = (char *)malloc( m * sizeof(char));
```

Естественно, что в этом случае длина строки может быть переменной и задаваться на этапе выполнения программы. Динамические строки, как и другие динамические массивы, нельзя инициализировать при создании. Оператор

```
char *str = "Never trouble trouble"
```

создает не строковую переменную, а указатель на строковую константу, изменить которую невозможно.

Ввод-вывод строк

Для ввода-вывода строк используются как уже известные нам объекты cin и cout, так и функции, унаследованные из библиотеки C. Рассмотрим сначала первый способ:

```
#include <iostream.h>
int main(){
  const int n = 80;
  char s[n];
  cin >> s; cout << s << endl;
  return 0;
}
```

Как видите, строка вводится точно так же, как и переменные известных нам типов. Запустите программу и введите строку, состоящую из одного слова. Запустите программу повторно и введите строку из нескольких слов. Во втором случае выводится только первое слово. Это связано с тем, что ввод выполняется до первого пробельного символа (то есть пробела, знака табуляции или символа перевода строки '\n')[1].

Можно ввести слова входной строки в отдельные строковые переменные:

```
#include <iostream.h>
int main(){
  const int n = 80;
  char s[n], t[n], r[n];
  cin >> s >> t >> r; cout << s << endl << t << endl << r << endl;
  return 0;
}
```

[1] Если во вводимой строке больше символов, чем может вместить выделенная для ее хранения область, поведение программы не определено. Скорее всего, она завершится аварийно.

Если требуется ввести строку, состоящую из нескольких слов, в одну строковую переменную, используются *методы* getline или get класса istream, объектом которого является cin. Во второй части практикума мы изучим, что такое методы класса[1], а пока можно пользоваться ими как волшебным заклинанием, не вдумываясь в смысл. Единственное, что нам пока нужно знать, это синтаксис вызова метода — после имени объекта ставится точка, а затем пишется имя метода:

```
#include <iostream.h>
int main(){
  const int n = 80;
  char s[n];
  cin.getline(s, n);  cout << s << endl;
  cin.get(s, n);      cout << s << endl;
  return 0;
}
```

Метод getline считывает из входного потока n - 1 символов или менее (если символ перевода строки встретится раньше) и записывает их в строковую переменную s. Символ перевода строки[2] также считывается (удаляется) из входного потока, но не записывается в строковую переменную, вместо него размещается завершающий 0. Если в строке исходных данных более n - 1 символов, следующий ввод будет выполняться из той же строки, начиная с первого несчитанного символа.

Метод get работает аналогично, но оставляет в потоке символ перевода строки. В строковую переменную добавляется завершающий 0.

Никогда не обращайтесь к разновидности метода get с двумя аргументами два раза подряд, не удалив \n из входного потока. Например:

```
cin.get(s, n);                          // 1 - считывание строки
cout << s << endl;                      // 2 - вывод строки
cin.get(s, n);                          // 3 - считывание строки
cout << s << endl;                      // 4 - вывод строки
cin.get(s, n);                          // 5 - считывание строки
cout << s << endl;                      // 6 - вывод строки
cout << "Конец - делу венец" << endl;   // 7
```

При выполнении этого фрагмента вы увидите на экране первую строку, выведенную оператором 2, а затем завершающее сообщение, выведенное оператором 7. Какие бы прекрасные строки вы ни ввели с клавиатуры в надежде, что они будут прочитаны операторами 3 и 5, метод get в данном случае «уткнется» в символ \n, оставленный во входном потоке от первого вызова этого метода (оператор 1). В результате будут считаны и, соответственно, выведены на экран пустые строки (строки, содержащие 0 символов). А символ \n так и останется «торчать» во входном потоке. Возможное решение этой проблемы — удалить символ \n из входного потока путем вызова метода get без параметров, то есть после операторов 1 и 3 нужно вставить вызов cin.get().

[1] Синонимом термина «метод» является «функция-член класса».

[2] Символ перевода строки '\n' появляется во входном потоке, когда вы нажимаете клавишу Enter.

Однако есть и более простое решение — использовать в таких случаях метод getline, который после прочтения строки не оставляет во входном потоке символ \n.

Если в программе требуется ввести несколько строк, метод getline удобно использовать в заголовке цикла, например:

```
#include <iostream.h>
int main(){
  const int n = 80;
  char s[n];
  while (cin.getline(s, n)) {
    cout << s << endl;
    …                  // обработка строки
  }
  return 0;
}
```

Рассмотрим теперь способы ввода-вывода строк, перекочевавшие в C++ из языка C. Во-первых, можно использовать для ввода строки известную нам функцию scanf, а для вывода — printf, задав спецификацию формата %s:

```
#include <stdio.h>
int main(){
  const int n = 10;
  char s[n];
  scanf("%s", s);  printf("%s", s);
  return 0;
}
```

Имя строки, как и любого массива, является указателем на его начало, поэтому использовавшаяся в предыдущих примерах применения функции scanf операция взятия адреса (&) опущена. Ввод будет выполняться так же, как и для классов ввода-вывода — до первого пробельного символа. Чтобы ввести строку, состоящую из нескольких слов, используется спецификация %c (символы) с указанием максимального количества вводимых символов, например:

```
scanf("%10c", s);
```

Количество символов может быть только целой константой. При выводе можно задать перед спецификацией %s количество позиций, отводимых под строку:

```
printf("%15s", s);
```

Строка при этом выравнивается по правому краю отведенного поля. Если заданное количество позиций недостаточно для размещения строки, оно игнорируется, и строка выводится целиком. Спецификации формата описаны в Учебнике на с. 387, а сами функции семейства printf — на с. 411 и далее.

Библиотека содержит также функции, специально предназначенные для ввода-вывода строк: gets и puts. Предыдущий пример с использованием этих функций выглядит так:

```
#include <stdio.h>
int main(){
  const int n = 10;
  char s[n];
  gets(s);  puts(s);
  return 0;
}
```

Функция gets(s) читает символы с клавиатуры до появления символа новой строки и помещает их в строку s (сам символ новой строки в строку не включается, вместо него в строку заносится нуль-символ). Функция возвращает указатель на строку s, а в случае возникновения ошибки или конца файла — NULL.

Функция puts(s) выводит строку s на стандартное устройство вывода, заменяя завершающий 0 символом новой строки. Возвращает неотрицательное значение при успехе или EOF при ошибке.

Функциями семейства printf удобнее пользоваться в том случае, если в одном операторе требуется ввести или вывести данные различных типов. Если же работа выполняется только со строками, проще применять специальные функции для ввода-вывода строк gets и puts.

Операции со строками

Для строк не определена операция присваивания, поскольку строка является не основным типом данных, а массивом. Присваивание выполняется с помощью функций стандартной библиотеки или посимвольно «вручную» (что менее предпочтительно, так как чревато ошибками). Например, чтобы присвоить строке p строку a, можно воспользоваться функциями strcpy или strncpy:

```
char a[100] = "Never trouble trouble";
char *p = new char [m];
strcpy(p, a);
strncpy(p, a, strlen(a) + 1);
```

Для использования этих функций к программе следует подключить заголовочный файл <string.h>.

Функция strcpy(p, a) копирует все символы строки, указанной вторым параметром (a), включая завершающий 0, в строку, указанную первым параметром (p).

Функция strncpy(p, a, n) выполняет то же самое, но не более n символов, то есть числа символов, указанного третьим параметром. Если нуль-символ в исходной строке встретится раньше, копирование прекращается, а оставшиеся до n символы строки p заполняются нуль-символами. В противном случае (если n меньше или равно длине строки a) завершающий нуль-символ в p не добавляется.

Обе эти функции возвращают указатель на результирующую строку. Если области памяти, занимаемые строкой-назначением и строкой-источником, перекрываются, поведение программы не определено.

Функция strlen(a) возвращает фактическую длину строки a, не включая нуль-символ.

Программист должен сам заботиться о том, чтобы в строке-приемнике хватило места для строки-источника (в данном случае при выделении памяти значение переменной m должно быть больше или равно 100), и о том, чтобы строка всегда имела завершающий нуль-символ.

ВНИМАНИЕ

Выход за границы строки и отсутствие нуль-символа являются распространенными причинами ошибок в программах обработки строк.

Для преобразования строки в целое число используется функция atoi(str). Функция преобразует строку, содержащую символьное представление целого числа, в соответствующее целое число. Признаком конца числа служит первый символ, который не может быть интерпретирован как принадлежащий числу. Если преобразование не удалось, возвращает 0.

Аналогичные функции преобразования строки в длинное целое число (long) и в вещественное число с двойной точностью (double) называются atol и atof соответственно.

Пример применения функций преобразования:

```
char a[] = "10) Рост - 162 см, вес - 59.5 кг";
int num;
long height;
double weight;
num  = atoi(a);
height = atol(&a[11]);
weight  = atof(&a[25]);
cout << num << ' ' << height << ' ' << weight;
```

Библиотека предоставляет также различные функции для сравнения строк и подстрок, объединения строк, поиска в строке символа и подстроки и выделения из строки лексем. Эти функции описаны в Учебнике на с. 414–446. В процессе разбора задач мы рассмотрим некоторые из них.

Работа с символами

Для хранения отдельных символов используются переменные типа char. Их ввод-вывод также может выполняться как с помощью классов ввода-вывода, так и с помощью функций библиотеки.

При использовании классов ввод-вывод осуществляется как с помощью операций помещения в поток << и извлечения из потока >>, так и методов get() и get(char). Ниже приведен пример применения операций:

```
#include <iostream.h>
int main(){
  char c, d, e;
  cin >> c;
  cin >> d >> e;
  cout << c << d << e << endl;
  return 0;
}
```

Вводимые символы могут разделяться или не разделяться пробельными символами, поэтому таким способом ввести символ пробела нельзя. Для ввода любого символа, включая пробельные, можно воспользоваться методами get() или get(c):

```
#include <iostream.h>
int main(){
  char c, d, e;
  c = cin.get(); cin.get(d); cin.get(e);
  cout << c << d << e << endl;
  return 0;
}
```

Метод get() возвращает код извлеченного из потока символа или EOF, а метод get(c) записывает извлеченный символ в переменную, переданную ему в качестве аргумента, а возвращает ссылку на поток.

В заголовочном файле <stdio.h> определена функция getchar() для ввода символа со стандартного ввода, а также putchar() для вывода:

```
#include <stdio.h>
int main(){
  char c, d;
  c = getchar();  putchar(c);
  d = getchar();  putchar(d);
  return 0;
}
```

В библиотеке также определен целый ряд функций, проверяющих принадлежность символа какому-либо множеству, например множеству букв (isalfa), разделителей (isspace), знаков пунктуации (ispunct), цифр (isdigit) и т. д. Описание этих функций приведено в Учебнике на с. 92 и с. 409–446.

Перейдем теперь к рассмотрению задач.

Задача 5.1. Поиск подстроки

Написать программу, которая определяет, встречается ли в заданном текстовом файле заданная последовательность символов. Длина строки текста не превышает 80 символов, текст не содержит переносов слов, последовательность не содержит пробельных символов.

На предыдущем семинаре на примере задачи 4.3 мы рассмотрели общий порядок действий при создании программы. Будем придерживаться его и впредь.

I. *Исходные данные и результаты*

Исходные данные:

1. Текстовый файл неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно огра-

ничиться поиском заданной последовательности в каждой строке отдельно. Следовательно, необходимо помнить только одну текущую строку файла. Для ее хранения выделим строковую переменную длиной 81 символ (дополнительный символ требуется для завершающего нуля).

2. Последовательность символов для поиска, вводимая с клавиатуры. Поскольку по условию задачи она не содержит пробельных символов, ее длина также не должна быть более 80 символов, иначе поиск завершится неудачей. Для ее хранения также выделим строковую переменную длиной 81 символ.

Результатом работы программы является сообщение либо о наличии заданной последовательности, либо об ее отсутствии. Представим варианты сообщений в программе в виде строковых констант.

Для хранения длины строки будем использовать именованную константу. Для работы с файлом потребуется служебная переменная соответствующего типа.

II. *Алгоритм решения задачи*

1. Построчно считывать текст из файла.
2. Для каждой строки проверять, содержится ли в ней заданная последовательность.
3. Если да, напечатать сообщение о наличии заданной последовательности и завершить программу.
4. При нормальном выходе из цикла напечатать сообщение об отсутствии заданной последовательности и завершить программу.

III. *Программа и тестовые примеры*

```
#include <fstream.h>
#include <string.h>
int main(){
  const int len = 81;                                          // 1
  char word[len], line[len];                                   // 2
  cout << "Введите слово для поиска: "; cin >> word;

  ifstream fin("text.txt", ios::in | ios::nocreate);           // 3
  if (!fin)  { cout << "Ошибка открытия файла." << endl;
    return 1; }                                                // 4

  while (fin.getline(line, len)) {                             // 5
    if (strstr(line, word)) {                                  // 6
      cout << "Присутствует!" << endl; return 0; }
  }
  cout << "Отсутствует!" << endl;
  return 0;                                                    // 7
}
```

Рассмотрим помеченные операторы. В операторе 1 описывается константа, определяющая длину строки файла и длину последовательности. В операторе 2 описывается переменная `line` для размещения очередной строки файла и переменная `word` для размещения искомой последовательности символов.

В операторе 3 определяется объект `fin` класса входных потоков `ifstream`. С этим объектом можно работать так же, как со стандартными объектами `cin` и `cout`, то есть использовать операции помещения в поток << и извлечения из потока >>, а также рассмотренные выше функции `get`, `getline` и другие. Предполагается, что файл с именем `text.txt` находится в том же каталоге, что и текст программы, иначе следует указать полный путь, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение, например:

```
ifstream fin("c:\\prim\\cpp\\text.txt", ios::in | ios::nocreate);  // 3
```

В операторе 4 проверяется успешность создания объекта `fin`. Файлы, открываемые для чтения, проверять нужно обязательно! В операторе 5 организуется цикл чтения из файла в переменную `line`. Метод `getline`, описанный выше, при достижении конца файла вернет значение, завершающее цикл.

Для анализа строки в операторе 6 применяется функция `strstr(line, word)`. Она выполняет поиск подстроки `word` в строке `line`. Обе строки должны завершаться нуль-символами. В случае успешного поиска функция возвращает указатель на найденную подстроку, в случае неудачи — `NULL`. Если вторым параметром передается указатель на строку нулевой длины, функция возвращает указатель на начало строки `line`.

В качестве тестового примера приготовьте текстовый файл, состоящий из нескольких строк[1]. Длина хотя бы одной из строк должна быть равна 80 символам. Для тестирования программы следует запустить ее по крайней мере два раза: введя с клавиатуры слово, содержащееся в файле, и слово, которого в нем нет.

Даже такую простую программу мы рекомендуем вводить и отлаживать по шагам. Это умение пригодится вам в дальнейшем. Предлагаемая последовательность отладки:

1. Ввести «скелет» программы (директивы #include, функцию `main()`, операторы 1–4). Добавить контрольный вывод введенного слова. Запустив программу, проверить ввод слова и успешность открытия файла. Выполнить программу, задав имя несуществующего файла, для проверки вывода сообщения об ошибке. Удалить контрольный вывод слова.
2. Проверить цикл чтения из файла: добавить оператор 5 с его завершающей фигурной скобкой, внутри цикла поставить контрольный вывод прочитанной строки:

   ```
   cout << line << endl;
   ```

   Удалить контрольный вывод строки.
3. Дополнить программу операторами проверки и вывода сообщений. Для полной проверки программы следует выполнить ее для нескольких последователь-

[1] Файл можно создать в любом текстовом редакторе, в том числе и в той оболочке, в которой вы работаете. Для правильного отображения русских букв при выводе на консоль вид кодировки должен быть ASCII.

ностей. Длина одной из них должна составлять максимально допустимую — 80 символов.

СОВЕТ

При вводе текста программы не ленитесь сразу же форматировать его и снабжать комментариями.

Задача 5.2. Подсчет количества вхождений слова в текст

Написать программу, которая определяет, сколько раз встретилось заданное слово в текстовом файле, длина строки в котором не превышает 80 символов. Текст не содержит переносов слов.

На первый взгляд эта программа не сильно отличается от предыдущей: вместо факта наличия искомой последовательности в файле требуется подсчитать количество вхождений слова, то есть после первого удачного поиска не выходить из цикла просмотра, а увеличить счетчик и продолжать просмотр. В целом это верно, однако в данной задаче нам требуется найти не просто последовательность символов, а законченное слово.

Определим слово как последовательность алфавитно-цифровых символов, после которых следует знак пунктуации, разделитель или признак конца строки. Слово может находиться либо в начале строки, либо после разделителя или знака пунктуации. Это можно записать следующим образом (фигурные скобки и вертикальная черта означают выбор из альтернатив):

```
слово =
  {начало строки | знак пунктуации | разделитель}
  символы, составляющие слово
  {конец строки | знак пунктуации | разделитель}
```

I. *Исходные данные и результаты*

Исходные данные:

1. Текстовый файл неизвестного размера, состоящий из строк длиной не более 80 символов. Поскольку по условию переносы отсутствуют, можно ограничиться поиском слова в каждой строке отдельно. Для ее хранения выделим строку длиной 81 символ.
2. Слово для поиска, вводимое с клавиатуры. Для его хранения также выделим строку длиной 81 символ.

Результатом работы программы является количество вхождений слова в текст. Представим его в программе в виде целой переменной.

Для хранения длины строки будем использовать именованную константу, а для хранения фактического количества символов в слове — переменную целого типа. Для работы с файлом потребуется служебная переменная соответствующего типа.

II. *Алгоритм решения задачи*

1. Построчно считывать текст из файла.
2. Просматривая каждую строку, искать в ней заданное слово. При каждом нахождении слова увеличивать счетчик.

Детализируем второй пункт алгоритма. Очевидно, что слово может встречаться в строке многократно, поэтому для поиска следует организовать цикл просмотра строки, который будет работать, пока происходит обнаружение в строке последовательности символов, составляющих слово.

При обнаружении совпадения с символами, составляющими слово, требуется определить, является ли оно отдельным словом, а не частью другого[1]. Например, мы задали слово «кот». Эта последовательность символов содержится, например, в словах «котенок», «трикотаж», «трескотня» и «апперкот». Следовательно, требуется проверить символ, стоящий после слова, а в случае, когда слово не находится в начале строки — еще и символ перед словом. Эти символы проверяются на принадлежность множеству знаков пунктуации и разделителей.

III. *Программа и тестовые примеры*

Разобьем написание программы на последовательность шагов.

Шаг 1. Ввести «скелет» программы (директивы #include, функцию main(), описание переменных, открытие файла). Добавить контрольный вывод введенного слова. Запустив программу, проверить ввод слова и успешность открытия файла. Для проверки вывода сообщения об ошибке следует выполнить программу еще раз, задав имя несуществующего файла.

```
#include <fstream.h>
int main(){
  const int len = 81;
  char word[len], line[len];
  cout << " Введите слово для поиска: "; cin >> word;
  ifstream fin("text.txt", ios::in | ios::nocreate);
  if (!fin) { cout << " Ошибка открытия файла." << endl; return 1; }
  return 0;
}
```

Шаг 2. Добавить в программу цикл чтения из файла, внутри цикла поставить контрольный вывод считанной строки (добавляемые операторы помечены признаком комментария):

```
#include <fstream.h>
int main(){
  const int len = 81;
  char word[len], line[len];
  cout << " Введите слово для поиска: "; cin >> word;
  ifstream fin("text.txt", ios::in | ios::nocreate);
  if (!fin) { cout << " Ошибка открытия файла." << endl;  return 1; }
```

[1] Кроме этого, слово может быть написано в разных регистрах, но мы для простоты будем искать точное совпадение.

```
    while (fin.getline(line, len)) {                                    //
    cout << line << endl;                                               //
  }                                                                     //
  return 0;
}
```

Шаг 3. Добавить в программу цикл поиска последовательности символов, составляющих слово, с контрольным выводом:

```
#include <fstream.h>
#include <string.h>                                                     //
int main(){
  const int len = 81;
  char word[len], line[len];
  cout << "Введите слово для поиска: "; cin >> word;
  int l_word = strlen(word);                                            //

  ifstream fin("text.txt", ios::in | ios::nocreate);
  if (!fin) { cout << "Ошибка открытия файла." << endl; return 1; }

  int count = 0;                                                        //
  while (fin.getline(line, len)) {
    char *p = line;                                                     //
    while( p = strstr(p, word)) {                                       //
      cout << " совпадение: " << p << endl;                             //
      p += l_word;  count++;                                            //
    }                                                                   //
  }
  cout << count << endl;                                                //
  return 0;
}
```

Для многократного поиска вхождения подстроки в заголовке цикла используется функция strstr. Очередной поиск должен выполняться с позиции, следующей за найденной на предыдущем проходе подстрокой. Для хранения этой позиции определяется вспомогательный указатель p, который на каждой итерации цикла наращивается на длину подстроки. Также вводится счетчик количества совпадений. На данном этапе он считает не количество слов, а количество вхождений последовательности символов, составляющих слово.

Шаг 4. Добавить в программу анализ принадлежности символов, находящихся перед словом и после него, множеству знаков пунктуации и разделителей:

```
#include <fstream.h>
#include <string.h>
#include <ctype.h>                                                      //
int main(){
  const int len = 81;
 char word[len], line[len];
  cout << " Введите слово для поиска: "; cin >> word;
  int l_word = strlen(word);

  ifstream fin("text.txt", ios::in | ios::nocreate);
  if (!fin) { cout << "Ошибка открытия файла." << endl; return 1; }

  int count = 0;
  while (fin.getline(line, len)) {
    char *p = line;
    while( p = strstr(p, word)) {
      char *c = p;                                                      //
      p += l_word;
      // слово не в начале строки?
      if (c != line)                                                    //
      // символ перед словом не разделитель?
        if ( !ispunct(*(c - 1) ) && !isspace(*(c - 1) )) continue;      //
      // символ после слова разделитель?
      if ( ispunct(*p) || isspace(*p) || (*p == '\0') )   count++;      //
    }
  }
  cout << "Количество вхождений слова: " << count << endl;
  return 0;
}
```

Здесь вводится служебная переменная c для хранения адреса начала вхождения подстроки. Символы, ограничивающие слово, проверяются с помощью функций ispunct и isspace, прототипы которых хранятся в заголовочном файле <ctype.h>. Символ, стоящий после слова, проверяется также на признак конца строки (для случая, когда искомое слово находится в конце строки).

Для тестирования программы требуется создать файл с текстом, в котором заданное слово встречается:

- в начале строки;
- в конце строки;
- в середине строки;
- несколько раз в одной строке;
- как часть других слов, находящаяся в начале, середине и конце этих слов;
- в скобках, кавычках и других разделителях.

Длина хотя бы одной из строк должна быть равна 80 символам. Для тестирования программы следует выполнить ее по крайней мере два раза: введя с клавиатуры слово, содержащееся в файле, и слово, которого в нем нет.

Давайте теперь рассмотрим другой вариант решения этой задачи. В библиотеке есть функция strtok, которая разбивает переданную ей строку на лексемы в соответствии с заданным набором разделителей. Если мы воспользуемся этой функцией, нам не придется «вручную» выделять и проверять начало и конец слова,

потребуется лишь сравнить с искомым словом слово, выделенное с помощью strtok. Правда, список разделителей придется задать вручную:

```
#include <fstream.h>
#include <string.h>
int main(){
  const int len = 81;
  char word[len], line[len];
  char delims[] = ",.!? /<>|)(*:;\"";
  cout << "Введите слово для поиска: "; cin >> word;

  ifstream fin("text.txt", ios::in | ios::nocreate);
  if (!fin) { cout << "Ошибка открытия файла." << endl; return 1; }

  char *token;
  int count = 0;
  while (fin.getline(line, len)) {
    token = strtok( line, delims );                          // 1
    while( token != NULL ) {
      if ( !strcmp (token, word) )count++;                   // 2
      token = strtok( NULL, delims );                        // 3
    }
  }
  cout << "Количество вхождений слова: " << count << endl;
  return 0;
}
```

Первый вызов функции strtok в операторе 1 формирует адрес первой лексемы (слова) строки line. Он сохраняется в переменной token. Функция strtok заменяет на NULL разделитель, находящийся после найденного слова, поэтому в операторе 2 можно сравнить на равенство искомое и выделенное слово. В операторе 3 выполняется поиск следующей лексемы в той же строке. Для этого следует задать в функции strtok в качестве первого параметра NULL.

Как видите, программа стала короче и яснее. На этом примере можно видеть, что средства, предоставляемые языком, влияют на алгоритм решения задачи, и поэтому перед тем, как продумывать алгоритм, необходимо эти средства изучить. Представьте, во что бы вылилась программа без использования функций работы со строками и символами!

Задача 5.3. Вывод вопросительных предложений

Написать программу, которая считывает текст из файла и выводит на экран только вопросительные предложения из этого текста.

I. *Исходные данные и результаты*

Исходные данные: текстовый файл неизвестного размера, состоящий из неизвестного количества предложений. Предложение может занимать несколько строк, поэтому ограничиться буфером на одну строку в данной задаче нельзя. Примем решение выделить буфер, в который поместится весь файл.

Результаты являются частью исходных данных, поэтому дополнительного пространства под них выделять не требуется.

Будем хранить длину файла в переменной длинного целого типа. Для организации вывода предложений понадобятся переменные того же типа, хранящие позиции начала и конца предложения.

II. *Алгоритм решения задачи*

1. Открыть файл.
2. Определить его длину в байтах.
3. Выделить в динамической памяти буфер соответствующего размера.
4. Считать файл с диска в буфер.
5. Анализируя буфер посимвольно, выделять предложения. Если предложение оканчивается вопросительным знаком, вывести его на экран.

Детализируем последний пункт алгоритма. Для вывода предложения необходимо хранить позиции его начала и конца. Предложение может оканчиваться точкой, восклицательным или вопросительным знаком. В первых двух случаях предложение пропускается. Это выражается в том, что значение позиции начала предложения обновляется. Оно устанавливается равным символу, следующему за текущим, и просмотр продолжается. В случае обнаружения вопросительного знака предложение выводится на экран, после чего также устанавливается новое значение позиции начала предложения.

III. *Программа и тестовые примеры*

Ниже приводится текст программы. Рекомендуем вам самостоятельно разбить его для отладки на последовательность шагов аналогично предыдущим примерам, вставляя и удаляя отладочную печать. Файл с тестовым примером должен содержать предложения различной длины (от нескольких символов до нескольких строк), в том числе и вопросительные.

```
#include <fstream.h>
#include <stdio.h>
int main(){
  ifstream fin("text.txt", ios::in | ios::nocreate);
  if (!fin) { cout << "Ошибка открытия файла." << endl; return 1; }

  fin.seekg(0, ios::end);                                    // 1
  long len = fin.tellg();                                    // 2
  char *buf = new char [len + 1];                            // 3
  fin.seekg(0, ios::beg);                                    // 4
  fin.read(buf, len);                                        // 5
  buf[len] = '\0';
  long n = 0, i = 0, j = 0;                                  // 6
  while(buf[i]) {                                            // 7
    if( buf[i] == '?') {                                     // 8
```

```
      for ( j = n; j <= i; j++) cout << buf[j];
      n = i + 1;
    }
    if ( buf[i] == '.' || buf[i] == '!' ) n = i + 1;
    i++;
  }
  fin.close();                                                    // 9
  cout << endl;
  return 0;
}
```

Для определения длины файла используются методы seekg и tellg класса ifstream. С любым файлом при его открытии связывается так называемая текущая позиция чтения или записи. Когда файл открывается для чтения, эта позиция устанавливается на начало файла. Для определения длины файла мы перемещаем ее на конец файла с помощью метода seekg (оператор 1), а затем с помощью tellg получаем ее значение, запомнив его в переменной len (оператор 2).

Метод seekg(offset, org) перемещает текущую позицию чтения из файла на offset байтов относительно org. Параметр org может принимать одно из трех значений:

ios::beg — от начала файла;

ios::cur — от текущей позиции;

ios::end — от конца файла.

beg, cur и end являются константами, определенными в классе ios, предке ifstream, а символы :: означают операцию доступа к этому классу.

В операторе 3 выделяется len + 1 байтов под символьную строку buf, в которой будет храниться текст из файла. Мы выделяем на один байт больше, чем длина файла, чтобы после считывания файла записать в этот байт нуль-символ.

Для чтения информации требуется снова переместить текущую позицию на начало файла (оператор 4). Собственно чтение выполняется в операторе 5 с помощью метода read(buf, len), который считывает из файла len символов (или менее, если конец файла встретится раньше) в символьный массив buf.

В операторе 6 определяются служебные переменные. В переменной n будет храниться позиция начала текущего предложения, переменная i используется для просмотра массива, переменная j — для вывода предложения.

Цикл просмотра массива buf (оператор 7) завершается, когда встретился нуль-символ. Если очередным символом оказался вопросительный знак (оператор 8), выполняется вывод символов, начиная с позиции n до текущей, после чего в переменную n заносится позиция начала нового предложения.

Оператор 9 (закрытие потока) в данном случае не является обязательным, так как явный вызов close() необходим только тогда, когда требуется закрыть поток раньше окончания действия его области видимости.

Если требуется вывести результаты выполнения программы не на экран, а в файл, в программе следует описать объект класса выходных потоков ofstream, а затем использовать его аналогично другим потоковым объектам, например:

```
ofstream fout("textout.txt");
if (!fout) { cout << "Ошибка открытия файла вывода" << endl; return 1; }
...
fout << buf[j];
```

Если требуется закрыть поток раньше окончания действия его области видимости, используется метод close:

```
fout.close();
```

При выполнении некоторых заданий этого семинара может потребоваться посимвольное чтение из файла. При использовании потоков оно выполняется с помощью метода get(). Например, для программы, приведенной выше, посимвольный ввод выглядит следующим образом:

```
while((buf[i] = fin.get()) != EOF) {
  ...
  i++;
}
```

Надо учитывать, что посимвольное чтение из файла гораздо менее эффективно.

В заключение приведем вариант решения этой же задачи с использованием вместо потоковых классов библиотечных функций, унаследованных из языка C.

```
#include <stdio.h>
int main(){
  FILE *fin;                                                      // 1
  fin = fopen("text.txt", "r");                                   // 2
  if (!fin) { puts("Ошибка открытия файла"); return 1; }

  fseek(fin, 0, SEEK_END);                                        // 3
  long len = ftell(fin);                                          // 4
  char  *buf = new char [len + 1];

  const int l_block = 1024;                                       // 5
  int num_block = len / l_block;                                  // 6
  fseek(fin, 0, SEEK_SET);                                        // 7
  fread(buf, l_block, num_block + 1, fin);                        // 8
  buf[len] = '\0';

  long n = 0, i = 0, j = 0;
  while(buf[i]) {
    if( buf[i] == '?') {
      for ( j = n; j <= i; j++) putchar(buf[j]);
      n = i + 1;
    }
    if ( buf[i] == '.' || buf[i] == '!' ) n = i + 1;
    i++;
  }
```

```
    fclose(fin);
    printf("\n");
    return 0;
}
```

В операторе 1 определяется указатель на описанную в заголовочном файле <stdio.h> структуру FILE. Указатель именно такого типа формирует функция открытия файла fopen. Ее вторым параметром задается режим открытия файла. В данном случае файл открывается для чтения (r).

Файл можно открыть в двоичном (b) или текстовом (t) режиме. Эти символы записывают во втором параметре, например, "rb" или "rt". Двоичный режим означает, что символы перевода строки и возврата каретки (0x13 и 0x10) обрабатываются точно так же, как и остальные. В текстовом режиме эти символы преобразуются в одиночный символ перевода строки. По умолчанию файлы открываются в текстовом режиме.

Для позиционирования указателя текущей позиции используется функция fseek с параметрами, аналогичными соответствующему методу потока (операторы 3 и 7). Константы, задающие точку отсчета смещения, описаны в заголовочном файле <stdio.h> и имеют имена:

SEEK_SET — от начала файла;

SEEK_CUR — от текущей позиции;

SEEK_END — от конца файла.

Чтение из файла выполняется функцией fread(buf, size, num, file) блоками по size байт. Требуется также задать количество блоков num. В программе размер блока задан в переменной l_block равным 1024, поскольку размер кластера кратен степени двойки. В общем случае чем более длинными блоками мы читаем информацию, тем быстрее будет выполнен ввод. Для того чтобы обеспечить чтение всего файла, к количеству блоков добавляется 1 для округления после деления.

Вывод на экран выполняется посимвольно с помощью функции putchar.

Если требуется с помощью функций библиотеки вывести результаты выполнения программы не на экран, а в файл, в программе следует описать указатель на структуру FILE, с помощью функции fopen открыть файл для записи (второй параметр функции — w), а затем использовать этот указатель в соответствующих функциях вывода, например:

```
FILE *fout;
fout = fopen("textout.txt", "w");
if (!fout) { puts("Ошибка открытия файла вывода"); return 1; }
…
putc(buf[j], fout);     // или fputc(buf[j], fout);
```

После окончания вывода файл закрывается с помощью функции fclose:

```
fclose(fout);
```

Функции вывода в файл описаны в Учебнике на с. 90 и 411.

Смешивать в одной программе ввод-вывод с помощью потоковых классов и с помощью функций библиотеки не рекомендуется.

В целом программа, написанная с использованием функций библиотеки, может получиться более быстродействующей, но менее безопасной, поскольку программист должен сам заботиться о большем количестве деталей.

Давайте повторим основные моменты этого семинара.

1. Длина динамической строки может быть переменной. Динамические строки нельзя инициализировать при создании.
2. Длина нединамической строки должна быть константным выражением.
3. При задании длины строки необходимо учитывать завершающий нуль-символ.
4. Присваивание строк выполняется с помощью функций библиотеки.
5. Для консольного ввода-вывода строк используются либо объекты cin и cout, либо функции библиотеки gets, scanf и puts, printf.
6. Ввод-вывод из файла может выполняться с помощью либо объектов классов ifstream и ofstream, либо функций библиотеки fgets, fscanf и fputs, fprintf.
7. Ввод строки с помощью операции >> выполняется до первого пробельного символа. Для ввода строки, содержащей пробелы, можно использовать либо методы getline или get класса istream, либо функции библиотеки gets и scanf.
8. Смешивать в одной программе ввод-вывод с помощью потоковых классов и с помощью функций библиотеки не рекомендуется.
9. Посимвольное чтение из файла неэффективно.
10. Разбивайте написание программы на последовательность шагов.
11. Выход за границы строки и отсутствие нуль-символа являются распространенными причинами ошибок в программах.
12. Средства, предоставляемые языком, влияют на алгоритм решения задачи, и поэтому перед тем, как продумывать алгоритм, необходимо эти средства изучить.
13. Программа, написанная с использованием функций, а не классов ввода-вывода, может получиться более быстродействующей, но менее безопасной.
14. Недостатком С-строк по сравнению с классом string является отсутствие проверки выхода строки за пределы отведенной ей памяти.