ЛАБОРАТОРНАЯ РАБОТА № 14
ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1. СТРОКИ

Строка – это массив символов (переменных типа **char**), который можно объявить, например, так:

```
char str[20];
```

Но строка – это не просто массив, строка должна заканчиваться нуль-символом **'\0'** (или просто кодом **0**), который еще называют концом строки. По положению нуль-символа определяется фактическая длина строки. Строку можно задать, инициализируя значения массива символов строковой константой. Строковая константа – это набор символов в двойных кавычках. Объявить строку и сразу ее проинициализировать можно так:

```
char str[20] = "Hello!";
```

или так:

```
char str[] = "Hello!";
```

В последнем примере под строку будет выделено 7 байт, 6 из которых занято под символы строки, а седьмой – под нуль-символ.

Можно объявить и проинициализировать строку отдельными символами:

```
char str[] = {'H', 'e', 'l', 'l', 'o', '!', '\0'};
```

Можно выделить память под строку динамически. Для этого в программе нужно объявить указатель на тип **char**:

```
char *Str; // Указатель на будущую строку
int size;

printf("Введите размер строки: ");
scanf("%d", &size);
Str = new char[size+1];  // Выделение памяти под size
                         // (или меньше) символов,
                         // последний символ (size+1)
                         // должен быть кодом 0
```

Поскольку строка символов является массивом, нельзя присвоить одну строку другой, необходимо выполнить операцию присваивания последовательно для каждого элемента массива, например, если заданы строки

```
char str1[23] = "C++ language functions";
char str2[23];
```

то, простым присваиванием скопировать **str1** в **str2** нельзя:

```
// str2 = str1; НЕЛЬЗЯ так присваивать одну строку другой!!!
```

Точно так же нельзя сравнивать строки:

```
// if (str1 == str2) Нельзя так сравнивать строки!!!
```

Для выполнения копирования, сравнения и других операций со строками можно написать свои функции или использовать функции из *стандартной библиотеки языка С*. Для использования стандартных функций надо включить в

программу заголовочный файл **string.h**. Перед вызовом некоторых из этих функций необходимо **убедиться, что для копирования, перемещения и т.п. выделено достаточно памяти**.

2. ВВОД СТРОКИ С КЛАВИАТУРЫ

Для ввода одного слова с клавиатуры можно использовать уже известную функцию **scanf**:

```
char str1[31];
printf ("Введите слово"); // не более 30 символов
scanf("%s", str1);
```

Но она вводит строку только до первого пробела. Для ввода строки целиком, включая пробелы, лучше использовать функцию **gets**:

```
char* gets(char *str);
```

Описание

Функция **gets()** считывает символы с клавиатуры и помещает их в массив символов, на который указывает **str**. Символы считываются до тех, пока не встретится символ *перевода строки* (или *новой строки*) – **'\n'**. Символ **'\n'** заменяется нуль-символом (**'\0'**).

В случае успеха **gets()** возвращает **str**, в противном случае она возвращает **NULL**.

Количество символов, считываемых **gets()**, не ограничивается, поэтому необходимо следить за тем, чтобы не выйти за границы массива, на который указывает **str**.

Пример

```
char str1[31];
printf ("Введите строку"); // не более 30 символов
gets(str1);
```

В **Visual Studio** вместо функции **gets()** нужно использовать «безопасную» функцию **gets_s()**:

```
char* gets_s(char* str, size_t size);
```

Где **size** – максимальная длина строки, которая будет введена (тип **size_t** является синонимом **unsigned long**).

```
char str1[31];
printf ("Введите строку"); // не более 30 символов
gets_s(str1, 30);
```

3. ОПРЕДЕЛЕНИЕ ДЛИНЫ СТРОКИ

strlen – возвращает длину строки

Функция **strlen()** определяет, сколько символов содержится в переданной ей строке (не считая завершающий символ конца строки – **'\0'**).

```
size_t strlen(char *string);
```

Описание

Возвращает длину строки без завершающего символа **'\0'** .

Пример

```
char s[80] = "C language functions";
int len;
. . . . . .
len = strlen(s); // len будет равно 20
```

4. ФУНКЦИИ КОПИРОВАНИЯ СТРОК

Чтобы записать новое значение в строку или скопировать информацию из одной строки в другую используют функции копирования. При неправильном использовании эти функции могут вызвать серьезную ошибку, например, если произойдет **выход за границы массива**. Это бывает в том случае, если строка, в которую копируется информация, имеет недостаточный **размер** (под нее выделено мало место в памяти).

В копировании участвуют две строки, они называются «*источник*» (строка, **откуда** копируется информация) и «*приемник*» (**куда** она записывается или добавляется).

Простое копирование выполняет функция **strcpy()**.

strcpy – копирует строку s2 в s1

```
char* strcpy(char *s1, const char *s2);
```

Описание

Копирует строку **s2** в **s1**, *включая* завершающий символ **'\0'**. Перед использованием функции необходимо убедиться, что строка **s1** содержит достаточно места для размещения символов из строки **s2**. Функция возвращает **s1**.

Пример

```
char str1[40], str2[31];
printf ("Введите строку"); // не более 30 символов
gets_s(str1, 30);

strcpy(str2, str1);
printf("%s\n", str2);
```

Если нужно скопировать не всю строку, а только заданное количество символов, используется функция **strncpy()**. Эта функция подобна функции **strcpy()**, но кроме аргументов **s1** и **s2** имеет еще один, где и задается количество символов, которые надо скопировать. Важно помнить, что эта функция *НЕ записывает завершающий нуль*, а только копирует символы (в отличие от нее **strcpy()** всегда копирует завершающий нуль).

strncpy – копирует строку s2 в s1 (не более n символов)

```
char *strncpy(char *s1, const char *s2, size_t n);
```

Пример

```
char str1[40], str2[31];
printf ("Введите строку"); // не более 40 символов
gets_s(str1, 30);

strncpy(str2, str1, 30); // копируем не более 30 символов
printf("%s\n", str2);
```

Еще одна функция – **strcat**() позволяет проводить **конкатенацию** строк, т.е. добавлять строку-источник в конец строки-приемника (завершающий нуль записывается автоматически). Надо помнить, что приемник должен иметь достаточный размер, чтобы вместить обе исходных строки.

strcat – добавляет строку s2 в конец строки s1

```
char* strcat(char *s1, const char *s2);
```

Описание

Добавляет копию **s2** в конец строки **s1**. Перед использованием функции необходимо убедиться, что строка **s1** содержит достаточно места для размещения символов из строк **s1** и **s2**. Функция возвращает **s1**.

Пример

```
char s1[40] = "begin", s2[30] = "end";
strcat(s1, " ");
strcat(s1, s2);
printf("%s\n", s1);    // "begin end"
```

5. СРАВНЕНИЕ СТРОК

Для сравнения строк используют функцию **strcmp**.

strcmp – посимвольно сравнивает две строки

```
int strcmp(const char *s1, const char *s2);
```

Описание

Сравнивает посимвольно строки **s1** и **s2**. Сравнение происходит по кодам символов, поэтому функция различает строчные и заглавные буквы, т.к. они имеют разные коды. Если строки не равны, функция возвращает «разность» между первой и второй строкой, т. е. разность кодов первых отличающихся символов. Эти числа можно использовать для сортировки строк: если «разность» отрицательна, значит первая строка «меньше» второй, следовательно, стоит за ней в алфавитном порядке.

Возвращаемое значение

Функция возвращает **0**, если **s1** равно **s2**; отрицательное число, если **s1** меньше **s2**; положительное число, если **s1** больше **s2**.

Пример

```
char pass[] = "wind"; // правильный пароль
char s[80]; // вспомогательная строка
printf ("Введите пароль: ");
scanf("%s", s);
if (strcmp(pass, s))  // Если не 0
  printf ("Неверный пароль\n");
```

Иногда надо сравнить не всю строку, а только первые несколько символов. Для этого служит функция **strncmp**. Третий параметр этой функции – количество сравниваемых символов. Принцип работы такой же как у **strcmp**.

strcnmp – посимвольно сравнивает две строки (не более n символов)

```
int strncmp(const char *s1, const char *s2, size_t n);
```

6. ФУНКЦИИ ПОИСКА В СТРОКАХ

Когда говорят о поиске в строках, обычно рассматривают две задачи: найти первый заданный символ с начала (или с конца), или найти заданную подстроку (если она есть). Первую задачу выполняют функции **strchr** (поиск с начала строки) и **strrchr** (поиск с конца строки), а вторую – функция **strstr**.

Все эти функции возвращают указатель на найденный символ (или на первый символ найденной подстроки). Это значит, что переменная, в которую записывается это значение, должна быть объявлена *как указатель на символьную переменную*.

Параметры функций такие: первый – где искать (строка), второй – что искать (один символ для функций **strchr** и **strrchr** или строка для **strstr**). Чтобы получить номер символа с начала строки, надо вычесть из полученного указателя адрес начала массива. Если поиск завершился неудачно, функции возвращают **NULL.**

strchr – ищет символ ch в строке s

```
char *strchr(const char *s, int ch);
```

Описание

Ищет первое вхождение символа **ch** в строке **s**.

Возвращаемое значение

Возвращает указатель на найденный символ. Если символ не найден, возвращается **NULL**.

strrchr – ищет символ ch в строке s

```
char *strchr(const char *s, int ch);
```

Описание

Ищет последнее вхождение символа **ch** в строке **s**.

Возвращаемое значение

Возвращает указатель на найденный символ. Если символ не найден, возвращается **NULL**.

strstr – ищет подстроку s2 в строке s1

```
char *strstr(const char *s1, const char *s2);
```

Описание

Функция определяет, содержится ли строка **s2** в строке **s1**.

Возвращаемое значение

Функция возвращает указатель на первое вхождение строки **s2** в строку **s1** или **NULL**, если **s2** не была найдена.

Пример

```
main()
{
  char str1[] = "Это первая строка";
  char str2[] = "А это вторая строка",
  char *p;

  p = strchr(str1, 'о');  // Ищем первую букву о в строке str1
  if (p != NULL)
  {
    printf("Первая буква о: индекс %d\n", p - str1);
    p = strrchr(str1, 'о'); // Последнюю букву о в строке str1
    printf("Последняя буква о: индекс %d\n", p - str1);
  }
  p = strstr(str2, "это"); // Ищем подстроку “это” в строке str2
  if ( p != NULL )
    printf("Нашли это в %s", str2);
  else
    printf("\nНет слова это в %s", str2);
}
```

Пример

При чтении строк *из файла* с помощью функции **fgets** на конце остается символ перехода на новую строку **'\n'**. Чаще всего он не нужен и надо его удалить – поставить на его месте нуль (признак конца строки). Делается это так:

```
char s[80], *p;
...// Здесь вводится строка из текстового файла в s
p = strrchr (s, '\n'); // ищем символ '\n'
if (p != NULL) // если нашли,
  *p = '\0'; // записываем в это место '\0'
```

Часто возникает задача найти в строке какой-либо символ из некоторого набора символов. Для этого используется функция **strpbrk.**

strpbrk – ищет какой-либо из заданных строкой sq символов в строке s

```
char *strpbrk(const char *s, const char *sq);
```

Описание

Ищет первое вхождение какого-либо символа из строки **sq** в строке **s**. Возвращает указатель на найденный символ.

Возвращаемое значение

Возвращает указатель на найденный символ. Если символ не найден, возвращается **NULL**.

Пример

Посчитать количество предложений в тексте. Предложения могут заканчиваться точкой, восклицательным или вопросительным знаками, поэтому нужно посчитать количество этих символов в тексте. Для поиска и подсчета используем функцию **strpbrk.**

```
char sf[] = ".!?";  // Строка с символами, которые будем искать
char Str[200] = "Солнце встает! Догорает заря. Звезды погасли.
Это утро?";         // Анализируемая строка
char* s;  // Указатель на очередной найденный символ
int count; // Счетчик предложений

s = Str;    // Указатель - на начало текста
count = 0;  // Счетчик равен 0
s = strpbrk (s, sf); // Ищем первое вхождение с начала текста
if (s != NULL) // Если нашли
  do
  {
    count++; // Увеличиваем счетчик
    s++;     // Сдвигаем указатель на следующий символ
    s = strpbrk (s, sf); // Ищем следующее вхождение символов
  } while (s != NULL); // пока они находятся

printf("Всего предложений: %d\n", count); // 4
```

7. ФОРМАТНЫЙ ВЫВОД В СТРОКУ

В программах часто требуется перед выводом информации сформировать всю строку для вывода целиком, включив в нее все необходимые данные. Например, сообщение об ошибке выводится стандартной функцией, и в это сообщение надо включить числовые данные. В этих случаях необходимо использовать функцию **sprintf**, которая поддерживает те же форматы данных, что и **printf**, но записывает результат не на экран, и не в файл, а *в символьную строку* (под нее надо заранее выделить память).

sprintf - форматный вывод в буфер

```
int sprintf(char *buffer, char *format, ...);
```

Описание

Функция работает точно так же как **printf**.

Пример

```
char s[80]; // вспомогательная строка
int x, y;

x = 1;
y = 5;
```

```
sprintf (s, "X=%d, Y=%d", x, y); // вывод в строку s
......  // В строке будет записано: "X=1, Y=5"
```

8. ФУНКЦИИ ПРЕОБРАЗОВАНИЯ К ВЕРХНЕМУ / НИЖНЕМУ РЕГИСТРАМ

strlwr – приведение к нижнему регистру

```
char *strlwr(char *str);
```

Описание

Функция преобразует все символы верхнего регистра строки **str** в соответствующие символы нижнего регистра.

Возвращаемое значение

Возвращается **str**.

strupr – приведение к верхнему регистру

```
char *strupr(char *str);
```

Описание

Функция преобразует все символы нижнего регистра строки **str** в соответствующие символы верхнего регистра.

Возвращаемое значение

Функция возвращает **str**.

Пример

```
char Str[] = "Abc Def";
. . . .
strlwr(Str);  // "abc def"
. . . .
strupr(Str);  // "ABC DEF"
```

9. ФУНКЦИИ ПРЕОБРАЗОВАНИЯ СТРОКИ В ЧИСЛО И ЧИСЛА В СТРОКУ

Для преобразования строки в число используются следующие функции:

int atoi(const char *s);

Описание

Преобразование строки **s** в целое десятичное число. Строка должна содержать корректную запись целого числа. Преобразование завершается на первом некорректном символе. Сюда относятся пробелы, знаки пунктуации и другие символы, не являющиеся цифрами. Таким образом, вызов функции **atoi()** для строки **"123.23"** возвратит целое значение (**123**), а часть **0.23** будет опущена. Начальные пробелы и знаки табуляции отбрасываются.

Возвращаемое значение

Возвращает целое число. Если строка некорректна, возвращается 0.

long atol(const char *s);

Описание

Преобразование строки **s** в длинное целое десятичное число. Строка должна содержать корректную запись целого числа. Преобразование завершается на первом некорректном символе. Сюда относятся пробелы, знаки пунктуации и другие знаки, не являющиеся цифрами. Таким образом, вызов функции **atol()** для строки **"123.23"** возвратит целое значение (**123**), а часть **0.23** будет опущена. Начальные пробелы и знаки табуляции отбрасываются.

Возвращаемое значение

Возвращает длинное целое число. Если строка некорректна, возвращается 0.

```
double atof(const char *s);
```

Описание

Преобразование строки **s** в число типа **double**. Строка должна содержать корректное число с плавающей точкой. Преобразование завершается на первом некорректном символе. Сюда относятся пробелы, знаки пунктуации, отличные от точки (запятой), буквы, отличные от «**E**» или «**e**». Таким образом, вызов функции **atof()** для строки **“123.456HELLO”** возвратит **123.456.** Начальные пробелы и знаки табуляции отбрасываются.

Возвращаемое значение

Возвращает число типа **double**. Если строка некорректна, возвращается 0.

Пример

```
k = atoi("  -123.456"); // k = -123
k = atoi("a123.456");  // k = 0
f = atof("123,456");   // f = 123,456000
f = atof("123.456");   // f = 123,000000
```

Для преобразования целого числа в строку используются следующие функции:

```
char* itoa(int val, char *s, int radix);
```

Описание

Функция **itoa()** конвертирует целое число **val** в строку. Результат записывается в строку, на которую указывает параметр **s**. В строке должно быть достаточно места для записи полученного выражения. Основание системы счисления для преобразования задается параметром **radix**, который может принимать значения в интервале от 2 до 36. В Visual Studio вместо функции **itoa()** нужно использовать «безопасную» функцию **_itoa()**.

Возвращаемое значение

Функция возвращает указатель на **str**.

Пример

```
char Str[80];
_itoa(127, Str, 10);  // “127”
_itoa(127, Str, 2);   // “1111111”
_itoa(127, Str, 16);  // “7f”
```

Аналогично работают функции для преобразования длинного целого и беззнакового длинного целого чисел в строку:

```
char *ltoa(long val, char *s, int radix);
```

```
char *ultoa(unsigned long val, char *s, int radix);
```

10. ФУНКЦИИ АНАЛИЗА СИМВОЛОВ

Для анализа отдельных символов текста удобны функции (точнее макросы), которые работают похожим образом: все они анализируют передаваемый в нее символ и возвращают ложь (0) или истину (не 0), в зависимости от результатов анализа. Данные макросы определены в файле **ctype.h**.

```
int isalnum(int c)
```

Описание

Функция возвращает не нуль, если **c** – код буквы или цифры, и нуль – в противном случае;

int isalpha(int c)

Описание

Функция возвращает не нуль, если **c** – код буквы, и нуль – в противном случае;

int isascii(int c)

Описание

Функция возвращает не нуль, если **c** есть код ASCII, т. е. код принимает значение от 0 до 127;

int isdigit(int c)

Описание

Функция возвращает не нуль, если **c** – цифра (**0–9**), и нуль – в противном случае;

int ispunct(int c)

Описание

Функция возвращает не нуль, если **c** – знак пунктуации, и нуль – в противном случае.

int isprint(int ch)

Описание

Функция возвращает не нуль, если **c** – печатный символ, включая пробел, и нуль – в противном случае. Значения печатных символов находятся в пределах от **0x20** до **0x7E**.

В файле **ctype.h** определены и другие подобные макросы.

11. ПРИМЕРЫ

Пример 1. *Подсчитать в тексте количество цифр*

Для решения подобной задачи проще всего просмотреть в цикле все символы текста и проанализировать их с помощью фукнции **isdigit** .

```
char Text[100] = "Today is January 31, 2022";
int main()
{
  setlocale(0, "Russian");
  int len = strlen(Text);
  int Count = 0;
  for (int i = 0; Text[i]; i++)
    if (isdigit(Text[i])) // Если очередной символ - цифра
      Count++;            // увеличиваем счетчик
  printf("%s", Text);
  printf("\nВсего символов – % d, из них цифр – % d", len, Count);
  return 0;
}
```

Для подсчета количества конкретных букв проще всего в цикле по очереди сравнивать все символы текста с нужной буквой:

```
  for (int i = 0; Text[i]; i++)
    if (Text[i] == 'a') // Если очередной символ – буква a
      Count++;          // увеличиваем счетчик
```

Хотя можно использовать функцию поиска **strchr**. Для этого нужно дополнительно объявить указатель на тип **char**, так как функция поиска возращает указатель на найденный символ. Но она возвращает указатель только на первый найденный символ, чтобы искать следующий, надо начинать его поиск со следующего за найденным символом. Для этого нужно сместить указатель на 1 и начинать новый поиск именно с него.

```
. . . .
int len = strlen(Text);
int Count = 0;
char* s = Text; // Указатель на очередной символ в строке
s = strchr(s, 'a'); // Ищем первое вхождение буквы ‘a’
if (s != NULL)  // Если нашли
do            // начинаем по очереди искать остальные
{
  Count++; // Счетчик найденных букв увеличиваем на 1
  s++;   // Сдвигаем указатель на следующий за найденным символ
  s = strchr(s, 'a'); // и ищем снова, начиная именно с него
} while (s != NULL);  // пока они находятся в строке

printf("\nВсего символов – % d, из них цифр – % d", len, Count);
```

Пример 2. *С клавиатуры вводится предложение и слово. Надо определить, сколько раз встречается это слово в предложении.*

Функция **strstr** может определить только первое вхождение слова в строке. Если нашли адрес первого данного слова в строке, нужно сохранить его в указатель **p**, а искать следующее слово нужно *не с начала строки*, а с адреса **p + длина_слова**. Повторяем эту операцию в цикле, пока функция **strstr** может найти слово в оставшейся части строки.

```
main()
{
  int len, count;
  char Text[80], word[20];
  char *p;       // указатель на найденное слово
  char *start;  // указатель на начало зоны поиска
  printf("Введите предложение");
  gets(Text);
  printf ("Введите слово для поиска");
  gets(word);
  len = strlen(word); // находим длину слова для поиска
  count = 0; // счетчик найденных слов
  start = Text; // в первый раз ищем с начала строки
  while (1)
  {
    p = strstr (start, word); // есть ли еще слова?
    if (p == NULL)
      break; // если нет, то выходим из цикла
    count++; // увеличить счетчик слов
    start = p + len; // сместили начало поиска
  }
  printf("В этом предложении %d слов %s", count, word);
}
```

Пример 3. *Выделить в тексте и вывести на печать слова, которые начинаются с гласной буквы.*

При решении подобных задач необходимо уметь проводить анализ текста: разделять текст на слова, определяя начало и конец слова, а также является ли начальная буква слова гласной. Разделять текст на слова можно по символам-разделителям, таким как пробел, запятая, точка и т.д. Можно искать разделители с

помощью функции **strpbrk**, по аналогии с предыдущим примером. А можно написать свою функцию, которая будет определять, является ли текущий символ разделителем, для чего необходимо сравнить его с этими символами. Напишем функцию, которая возвращает **1**, если символ является разделителем, и **0**, если нет:

```
int IsSeparator(char c) // Является ли символ 'c' разделителем
{
 char str[] = ",. !?:;:'\n\0"; // Строка с разделителями
 int i = 0;
 while (str[i] != 0) // До конца строки с разделителями
 {   // Сравниваем символ с по очереди со всеми разделителями
   if (c == str[i])
     return 1;  // Если совпадает, возвращаем 1
   i++;
 }
 return 0; // Не совпадает ни с одним из символов-разделителей
}
```

Чтобы определить является ли буква гласной, нужно написать похожую функцию, только строка, с которой будет сравниваться символ, должна состоять из гласных букв.

```
int IsVowel(char c) // Является ли символ гласной буквой
{
  char str[] = "aAeEyYuUiIoOjJaAyYeEыЫoOэЭяЯиИюЮ"; // гласные
  int i = 0;
  while (str[i] != 0) // До конца строки с разделителями
  {   // Сравниваем символ с по очереди со всеми разделителями
    if (c == str[i])
      return 1;  // Если совпадает, возвращаем 1
    i++;
  }
  return 0;  // Не совпадает ни с одной гласной буквой
}
```

Будем анализировать текст, заданный прямо в тексте программы строкой:

```
char Text[1024] = "Mark Twain is a famous American writer and "
" a journalist. His real name was Samuel Clemens. He was born "
" in 1835 in Florida. Mark Twain started his literary career "
" rather late. He worked as a journalist in newspapers in Nevada "
" and California during the years of the Civil War.";
```

Чтобы подсчитать количество слов, которые начинаются с гласной буквы, будем разделять его на отдельные слова по разделителям. Объявим указатель (**sr**) на анализируемый текст и еще один вспомогательный массив символов, куда будем записывать отдельные слова из текста (***st***).

Объявим вспомогательную переменную-флаг **flag**, с помощью которой будем следить за тем, началось ли выделение очередного слова из строки и сколько символов слова уже выделено.

```
int main()
{
  char ch = ' ';
  char* sr = Text;   // sr - Указатель на анализируемый текст
  char st[81]; // st - массив для сохранения слова из строки
  int flag = -1; // Слово еще не начало формироваться
```

```
  for (int i = 0; sr[i]; i++) // Начало цикла по i
  {
    ch = sr[i]; // очередной символ
    if (IsSeparator(ch)) // Если это символ-разделитель
    {
      if (flag != -1) // и слово уже начало формироваться
      {               // завершаем формировать слово:
        flag++; // Определяем индекс для симв. '\0' (конец строки)
        st[flag] = '\0'; // и ставим '\0' в конце st,
          // формируя тем самым строку (в ней скопированное слово)
              // Теперь в st лежит очередное слово из текста,
              // с которым можно что-либо делать, например
              // определить, первая буква слова гласная или нет:
        if (IsVowel(st[0])) // Если у слова 1-я буква гласная,
          printf("%s\n", st);    // то слово выводится на экран
        flag = -1; // Устанавливаем флаг в -1, чтобы готовиться
              // формировать следующее слово
      }
    }
    else // Если символ не разделитель - формируем слово
    {
      flag++; // Определяем индекс для текущего символа слова
      st[flag] = ch; // В массив для слова st пишем текущий символ
    }
  } // Конец цикла по i
  return 0;
}
```

Можно использовать для выделения слов стандартную функцию `strpbrk` и два указателя – на начало и конец слова. Сначала эти указатели указывают на начало текста. Затем ищем разделитель, т.е. конец слова. Если находим, вычисляем длину слова, как разность между указателем на конец и указателем на начало. Копируем в отдельный массив `st` слово, ставим символ конца строки, а затем анализируем скопированное слово и, если оно начинается с гласной, выводим его на дисплей.

```
  int main()
  {
    int slen, count = 0;
    char *sbeg, *send; // Указатели на начало и конец слова
    char st[81]; // st - для слова из строки
    char sf[] = " ,;:.!?"; // Разделители слов

    send = sbeg = Text; // На начало текста
    while (1)
    {
      send = strpbrk(sbeg, sf); // Ищем символ-разделитель
      if (send == NULL) break;  // Если нет разделителей - выходим
      slen = send - sbeg;       // Вычисляем длину слова
      strncpy(st, sbeg, slen); // Копируем в отельный массив
      st[slen] = 0; // Ставим конец строки
      if (IsVowel(st[0])) // Если у слова 1-я буква гласная,
        printf("%s\n", st); // Печатаем его
      send++; // Сдвигаем конец на следующий символ
      while (send[0] == ' ') send++; // Пропускаем пробелы
```

```
            sbeg = send; // Смещаем начало на него
            count++; // Увеличиваем счетчик слов (если надо)
        }
        return 0;
    }
```

Результат выполнения программы

```
is
a
American
and
a
journalist
in
in
as
a
journalist
in
in
and
years
of
```

После выделения слова можно анализировать его, например, определять длину слова, с какой буквы оно начинается, сколько в нем гласных букв и т.д.

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

в начале программы ОБЯЗАТЕЛЬНО выводить:
ФИО, группа, номер лаб. работы, номер варианта.

Задания

Написать программу анализа текста. В программе должно быть два варианта анализируемого текста: первый задается константной строкой прямо в тексте программы (как в вышеуказанных примерах), второй – вводится с клавиатуры, например, функцией ***gets()***. Программа должна позволять выбирать нужный текст для анализа. Вариант задания выдает преподаватель.

Для получения максимального балла нужно выполнить следующие требования:

- *Алгоритмы анализа должны быть реализованы с помощью функций.*
- *Программа должна содержать цикл для возможного повторного ввода и анализа текста.*

Варианты заданий

1. Выделить из текста слова размером от 2 до 10 символов и вывести эти слова на дисплей, а также количество символов в каждом из них.
2. Выделить из текста слова, которые начинаются с согласной буквы и вывести эти слова на дисплей, а также количество символов в каждом из них.
3. Выделить из текста все слова, которые начинаются и заканчиваются одной и той же буквой, и вывести эти слова на дисплей, а также количество символов в каждом из них.
4. Выделить из текста все слова, которые начинаются с заглавной буквы, и вывести эти слова на дисплей, а также количество символов в каждом из них.
5. Выделить из текста слова, которые заканчиваются гласной буквой и вывести эти слова на дисплей, а также количество символов в каждом из них.
6. Выделить из текста предложения и вывести их на дисплей, а также количество гласных букв в каждом из них.
7. Выделить из текста предложения и вывести их на дисплей, а также количество согласных букв в каждом из них.
8. Выделить из текста предложения и вывести их на дисплей, а также количество слов в каждом из них (можно определить по числу пробелов).
9. Выделить из текста все слова, которые начинаются с заглавной буквы, и вывести эти слова на дисплей, а также количество символов в каждом из них.
10. Выделить из текста слова размером более 3 символов и вывести эти слова на дисплей, а также количество символов в каждом из них.
11. Выделить из текста предложения и вывести их на дисплей, а также количество запятых в каждом из них.
12. Выделить из текста предложения и вывести их на дисплей, а также количество букв (без знаков пунктуации и пробелов) в каждом из них.
13. Выделить из текста все слова, которые содержат букву **a**, и вывести эти слова на дисплей, а также количество символов в каждом из них.
14. Выделить из текста слова, у которых вторая буква гласная, и вывести эти слова на дисплей, а также количество символов в каждом из них.
15. Ввести с клавиатуры слово. Выделить из текста предложения, в которых есть это слово, и вывести найденные предложения на дисплей, а также количество символов в каждом из них.
16. Подсчитать, сколько в тексте букв, сколько цифр, сколько знаков пунктуации, сколько пробелов и вывести информацию на дисплей.

17. Выделить из текста все числа, которые в нем содержатся, и вывести их на дисплей, а также количество цифр в каждом из них.

18. Подсчитать, сколько в тексте гласных букв, сколько согласных, сколько знаков пунктуации, сколько пробелов и вывести информацию на дисплей.

19. Выделить из текста последние слова всех предложений текста и вывести эти слова на дисплей, а также количество символов в каждом из них.

20. Проверить правильность написания знаков пунктуации. Слева от знака пунктуации пробел не ставится, справа ставится один пробел. Сохранить исправленный текст в отдельный массив. Вывести на дисплей исходный и исправленный тексты.

ПРИЛОЖЕНИЕ

Коды некоторых символов (таблица ASCII – American Standard Code for Information Interchange). Первые 32 символа таблицы ASCII (от 0x00 до 0x1F) использовались для непечатаемых символов. Они были предназначены для управления печатающим устройством и т.п. Остальная часть – от 0x20 до 0x7F – обычные (печатаемые) символы.

Коды от 0x80 до 0xFF определяют символы национальных алфавитов, псевдографики и т.д. и зависят от так называемой кодовой таблицы.

Табл.1. Непечатаемые символы

DEC	HEX	Символ	Escape послед.	Описание
0	0x00	NUL	\0	Нулевой байт
1	0x01	SOH		Начало заголовка
2	0x02	STX		Начало текста
3	0x03	ETX		Конец «текста»
4	0x04	EOT		конец передачи
5	0x05	ENQ		«Прошу подтверждения!»
6	0x06	ACK		«Подтверждаю!»
7	0x07	BEL	\a	Звуковой сигнал – звонок
8	0x08	BS	\b	Возврат на один символ (BACKSPACE)
9	0x09	TAB	\t	Табуляция
10	0x0A	LF	\n	Перевод строки
11	0x0B	VT	\v	Вертикальная табуляция
12	0x0C	FF	\f	Прогон страницы, новая страница
13	0x0D	CR	\r	Возврат каретки
14	0x0E	SO		Переключиться на другую ленту (кодировку)
15	0x0F	SI		Переключиться на исходную ленту (кодировку)
16	0x10	DLE		Экранирование канала данных
17	0x11	DC1		1-й символ управления устройством
18	0x12	DC2		2-й символ управления устройством
19	0x13	DC3		3-й символ управления устройством
20	0x14	DC4		4-й символ управления устройством
21	0x15	NAK		«Не подтверждаю!»
22	0x16	SYN		Символ для синхронизации
23	0x17	ETB		Конец текстового блока
24	0x18	CAN		Отмена
25	0x19	EM		Конец носителя
26	0x1A	SUB		Подставить
27	0x1B	ESC	\e	Escape
28	0x1C	FS		Разделитель файлов
29	0x1D	GS		Разделитель групп
30	0x1E	RS		Разделитель записей
31	0x1F	US		Разделитель юнитов
127	0x7F	Delete		Символ для удаления

Табл.2. Основная часть ASCII таблицы – обычные (печатаемые) символы

Dec	Hex	Символ
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Табл.3. Старшая половина ASCII таблицы кодов символов (кодировка 1251)

Dec	Hex	Символ
128	80	Ђ
129	81	Ѓ
130	82	‚
131	83	ѓ
132	84	„
133	85	…
134	86	†
135	87	‡
136	88	€
137	89	‰
138	8A	Љ
139	8B	‹
140	8C	Њ
141	8D	Ќ
142	8E	Ћ
143	8F	Џ
144	90	ђ
145	91	‘
146	92	’
147	93	“
148	94	”
149	95	•
150	96	–
151	97	—
152	98	□
153	99	™
154	9A	љ
155	9B	›
156	9C	њ
157	9D	ќ
158	9E	ћ
159	9F	џ
160	A0	
161	A1	Ў
162	A2	ў
163	A3	J
164	A4	¤
165	A5	Ґ
166	A6	¦
167	A7	§
168	A8	Ё
169	A9	©
170	AA	Є
171	AB	«
172	AC	¬
173	AD	-
174	AE	®
175	AF	Ї
176	B0	°
177	B1	±
178	B2	I
179	B3	i
180	B4	ґ
181	B5	µ
182	B6	¶
183	B7	·
184	B8	ё
185	B9	№
186	BA	є
187	BB	»
188	BC	j
189	BD	S
190	BE	s
191	BF	ї
192	C0	А
193	C1	Б
194	C2	В
195	C3	Г
196	C4	Д
197	C5	Е
198	C6	Ж
199	C7	З
200	C8	И
201	C9	Й
202	CA	К
203	CB	Л
204	CC	М
205	CD	Н
206	CE	О
207	CF	П
208	D0	Р
209	D1	С
210	D2	Т
211	D3	У
212	D4	Ф
213	D5	Х
214	D6	Ц
215	D7	Ч
216	D8	Ш
217	D9	Щ
218	DA	Ъ
219	DB	Ы
220	DC	Ь
221	DD	Э
222	DE	Ю
223	DF	Я
224	E0	а
225	E1	б
226	E2	в
227	E3	г
228	E4	д
229	E5	е
230	E6	ж
231	E7	з
232	E8	и
233	E9	й
234	EA	к
235	EB	л
236	EC	м
237	ED	н
238	EE	о
239	EF	п
240	F0	р
241	F1	с
242	F2	т
243	F3	у
244	F4	ф
245	F5	х
246	F6	ц
247	F7	ч
248	F8	ш
249	F9	щ
250	FA	ъ
251	FB	ы
252	FC	ь
253	FD	э
254	FE	ю
255	FF	я