Лабораторная работа №7
Обработка текстовых строк

Цель работы: получить навыки обработки текстовых строк с использованием языка C/C++.

Теоретические сведения

Представление символьной информации в ЭВМ

Символьная информация хранится и обрабатывается в компьютере в виде цифрового кода, то есть каждому символу ставится в соответствие некоторое число-код. Для разных типов компьютеров и операционных систем используются различные наборы символов. Необходимый набор символов, предусмотренный в том или ином компьютере, обычно включает:

- управляющие символы, соответствующие определенным функциям;
- цифры;
- буквы алфавита;
- специальные знаки (пробел, скобки, знаки препинания и т.д.);
- знаки операций.

Для представления символьной информации в компьютере чаще всего используется алфавит, состоящий из 256 символов. Каждый символ такого алфавита можно закодировать 8 битами (1 байтом) памяти. Все символы пронумерованы от 0 до 255, причем каждому номеру соответствует 8-разрядный двоичный код от 00000000 до 11111111.

Среди наборов символов наибольшее распространение получила таблица кодировки ASCII (*American Standard Code for Information Interchange* – американский стандартный код для обмена информацией). В данной таблице стандартными являются только первые 128 символов (символы с номерами от 0 до 127). Сначала размещаются управляющие (неотображаемые) символы (символы с номерами от 0 до 31). Далее идут буквы латинского алфавита, цифры, знаки препинания, скобки и некоторые другие символы, причем латинские буквы располагаются в алфавитном порядке, цифры также упорядочены от 0 до 9. Остальные 128 кодов таблицы символов ASCII используются для размещения символов национальных алфавитов, символов псевдографики и научных символов.

Для представления символьных переменных в языке Си используется тип char. Значением переменной типа char является 8-битный код, соответствующий тому или иному символу.

Библиотека обработки символов

Библиотека обработки символов (<ctype.h>) содержит функции, выполняющие проверки и операции с символьными данными. Каждая функция получает в качестве аргумента целое число, которое должно быть значением unsigned char либо EOF. Функции возвращают ненулевое значение (истина), если аргумент *c* удовлетворяет условию, и 0 (ложь) – иначе.

Прототип	Описание функции
int isdigit(int c)	проверка, является ли символ *c* десятичной цифрой
int islower(int c)	проверка, является ли символ *c* латинской буквой нижнего регистра (‘a’ – ‘z’)
int isupper(int c)	проверка, является ли *c* латинской буквой верхнего регистра (‘A’ – ‘Z’)
int isalpha(int c)	проверка, является ли *c* латинской буквой (isalpha(c) = islower(c) || isupper(c))
int isalnum(int c)	буква либо цифра (isalnum(c) = isalpha(c) || isdiglt (c))
int tolower(int c)	перевод латинского символа *c* на нижний регистр
int toupper(int c)	перевод латинского символа *c* на верхний регистр
int isspace(int c)	проверка, является ли *c* пробелом (‘ ’), сменой страницы (‘\f’), новой строкой (‘\n’), возвратом каретки (‘\r’), табуляцией (‘t’), вертикальной табуляцией (‘\v’)
int iscntrl(int c)	проверка, является ли *c* управляющим символом
int isprint(int c)	проверка, является ли *c* печатаемым символом, включая пробел
int isgraph(int c)	проверка, является ли *c* печатаемым символом, кроме пробела
int ispunct(int c)	проверка, является ли *c* печатаемым символом, кроме пробела, буквы или цифры

Например, после инструкции

```
printf("%s%s%s\n", "5 ", isdigit('5') ? "is " : "not is ", "a digit");
```

на экране появится следующая строка: 5 is a digit.

Строки в языке Си

В языке Си нет такого отдельного типа данных, как строка. Но можно определить строку двумя различными способами.

Строка как массив символов. Во-первых, строку можно определить как массив символов, который заканчивается нулевым символом ‘\0’. Нуль-символ (‘\0’) используется для того, чтобы отмечать конец строки. В таблице символов ASCII данный символ имеет номер 0. При определении строки как массива символов ей присваивается имя и указывается максимальное количество символов, которое может содержаться в ней с учетом нулевого символа. Например, char *s*[10]. Такое объявление строки в виде массива символов требует определенного места в памяти для десяти элементов. Значением строки *s* является адрес ее первого символа, через который осуществляется доступ ко всем ее элементам.

Например, в следующей программе объявляется массив символов *s*, который заполняется пользователем с помощью функции fgets(), а затем выводится на экран при помощи функции puts().

```
#include <stdio.h>
int main( )
{
  char s[10];       /* массив символов */
  printf("введите не более 9 символов: ");
  fgets(s, 10, stdin);
  puts(s);          /* вывод строки */
  return 0;
}
```

При объявлении строка может быть сразу же инициализирована, то есть она может быть присвоена массиву символов:

```
char s[] = “moon”;
```

При таком объявлении создается массив *s*, состоящий из 5 элементов: ‘m’, ‘o’, ‘o’, ‘n’, ‘\0’. Отдельные символы внутри массива могут изменяться, но сама переменная *s* будет указывать на одно и то же место памяти.

Объявление строки в виде массива символов с последующей инициализацией можно записать также следующим образом:

char s1[5] = {‘m’, ‘o’, ‘o’, ‘n’, ‘\0’};

char s2[7] = {‘m’, ‘o’, ‘o’, ‘n’, ‘\0’}.

После такого объявления массивы символов *s*1 и *s*2 будут иметь такой вид:

*s*1:

m	o	o	n	\0

*s*2:

m	o	o	n	\0	\0	\0

То же самое можно сделать и в более удобной форме, а именно:

char s1[5] = “moon”;

char s1[7] = “moon”;

Подчеркнем, что наличие нулевого символа означает, что количество элементов массива символов должно быть по крайней мере на один больше, чем максимальное количество символов, планируемых для размещения в памяти.

Заметим, что после объявления строки, например

char s[10];

абсолютно недопустима запись вида

s = “Hello”;

то есть имя массива нельзя использовать в левой части оператора присваивания.

Строка как указатель на первый символ. Строку можно также определить и другим способом – с использованием указателя на символ. Если объявить char **ps*, то этим самым задается переменная *ps*, которая может содержать адрес первого символа строки. В этом случае не происходит резервирования памяти для хранения символов, как в случае с массивом символов, и сама переменная *ps* не инициализируется конкретным значением.

Если где-то в программе после объявления char *ps встречается инструкция вида *ps* = “sun”, тогда компилятор производит следующие действия: в некоторую область памяти (скорее всего в область памяти *read only*) последовательно друг за другом записываются символы ‘s’, ‘u’, ‘n’ и ‘\0’; переменной *ps* присваивается адрес первого символа, то есть, в нашем случае, адрес символа ‘s’. После инициализации строки *ps* к каждому ее символу можно обращаться через индекс: *ps*[0], *ps*[1], *ps*[2].

Ниже приводится пример использования указателя на строку:

```
#include <stdio.h>
int main( )
{
  char s[17] = "this is a string";
  /* переменной ps присваивается адрес первого символа строки s */
  char *ps = s;
  int i;
  for (i = 0; i < 16; i++)
    /* обращение к элементам строки ps по их индексам: */
    printf("%c", ps[i]);
  printf("\n");
  ps = ps + 8;      /* указателю ps присваивается адрес буквы ‘a’ */
  printf("%s\n", ps);     /* вывод на экран новой строки */
  return 0;
}
```

В результате на экране появятся две строчки:

this is a string

a string

Как и в случае с массивом символов, строку, объявленную через указатель на первый символ, можно сразу же инициализировать, например

char *ps = “sun”;

В этом случае создается переменная-указатель *ps*, которая указывает на строку “sun”.

Заметим, что строка, определенная как указатель на первый символ строки, может быть помещена в неизменяемую область памяти, то есть значения элементов в этом случае изменить будет нельзя, хотя можно будет изменить значение самого указателя *ps*.

Еще раз подчеркнем, что при объявлении строки как массива символов (например, char *s*[10]) происходит резервирование места в памяти, в то время как объявление указателя (char *s) требует место только для самого указателя, который может указывать на любой символ или массив символов.

Ниже приводятся несколько примеров работы с символами и строками.

Пример 1. Преобразование строки *s* в целое число.

```
int Atoi(const char *s)
{
  int i, n;
  i = n = 0;
  while (s[i] >= '0' && s[i] <= '9')
  {
    n = n*10 + (s[i] - '0');
    i++;
  }
  return n;
}
```

В данном случае выражение *s*[*i*] – ‘0’ дает численное значение символа, который хранится в *s*[*i*], поскольку в таблице символов цифры ‘0’, ‘1’, …, ’9’ идут подряд. То есть ‘0’ преобразуется в 0, ‘1’ преобразуется в 1 и так до символа ‘9’, который преобразуется в число 9.

Пример 2. Преобразование прописного латинского символа в строчный. Если же символ не является прописной латинской буквой, то он остается без изменения.

```
int ToLower(const int c)
{
  if (c >= 'A' && c <= 'Z')
    return c + ('a' - 'A');
  else return c;
}
```

Пример 3. Функция удаления из строки *s* всех символов, совпадающих с символом *c*.

В функции DeleteSymbol() будем перезаписывать все символы, входящие в строку *s*, таким образом, чтобы среди них не оказался символ *c*.

```
void DeleteSymbol(char *s, const int c)
{
  int i, j;
  i = j = 0;
  while (s[i] != '\0')
  {
    if (s[i] != c)
      s[j++] = s[i];
    i++;
  }
  s[j] = '\0';
}
```

Пример 4. Определение длины строки без учета символа конца строки ‘\0’.

```
int Strlen(const char *s)
{
  int i = 0;
  while (s[i] != ‘\0’)
    i++;
  return i;
}
```

Пример 5. Копирование строки *t* в строку *s*.

```
void Strcpy(char *s, const char *t)
{
  int i = 0;
  while ((s[i] = t[i]) != '\0')
    i++;
}
```

Пример 6. Добавление строки *t* к строке *s*.

```
void Strcat(char *s, const char *t)
{
  int i, j;
  i = j = 0;
  while (s[i] != '\0')
    i++;
  while ((s[i++] = t[j++]) != '\0')
    ;
}
```

Функции обработки строк

Ниже приведены основные функции операций над строками (библиотека <string.h>):

Прототип	Описание функции
int strlen(const char *s)	возвращает длину строки *s*
char *strcpy(char *s, const char *t)	копирует строку *t* в строку *s*, включая ‘\0’; возвращает *s*
char *strncpy(char *s, const char *t, size_t n)	копирует не более *n* символов строки *t* в *s*; возвращает *s*. Дополняет результат символами ‘\0’, если символов в *t* меньше, чем *n*
char *strcat(char *s, const char *t)	приписывает *t* к *s*; возвращает *s*

char *strncat(char *s, const char *t, size_t n)	приписывает не более *n* символов *t* к *s*, возвращает *s*
int strcmp(const char *s, const char *t)	сравнивает *s* и *t*; возвращает <0, если *s*<*t*, 0, если *s* == *t*, и >0, если *s*>*t*
int strncmp(const char *s, const char *t, size_t n)	аналогична функции strcmp(), только сравниваются не более *n* первых символов в строках *s* и *t*
char *strchr(const char *s, int c)	возвращает указатель на первое вхождение символа *c* в строку *s*, либо NULL, если такого символа не оказалось
char *strrchr(const char *s, int c)	возвращает указатель на последнее вхождение символа *c* в строку *s*, либо NULL, если такого символа не оказалось
char *strpbrk(const char *s, const char *t)	возвращает указатель в строке *s* на первый символ, который совпал с одним из символов, входящих в *t*, либо NULL, если такового не оказалось
char *strstr(const char *s, const char *t)	возвращает указатель на первое вхождение строки *t* в строку *s*, либо NULL, если такой подстроки в *s* не оказалось
char *strtok(char *s, const char *t)	ищет в *s* лексему, ограниченную символами из *t*. Возвращает указатель на первый символ лексемы, либо NULL, если лексемы не существует

Приведем пример использования функции strchr(). После выполнения следующих строчек программы

```
char *s, c;
s = "check";
c = 'e';
```

```
printf("%s%c%s%s%s\n", "symbol ", c, strchr(s, c) ? " was " : " was not ",
"found in ", s);
```

на экране появится такой результат:

```
symbol e was found in check.
```

Рассмотрим также пример использования функции strstr():

```
char *s, *t;
s = "this is a test!!!";
 t = "a test";
printf("%s\n", strstr(s, t));
```

После выполнения данных строк на экране появится такой результат:

```
a test!!!.
```

Функция strtok() применяется для разбиения строки на лексемы, ограниченные разделительными символами (пробелы, знаки пунктуации). Первое обращение к strtok() возвращает значение указателя на первый символ первой лексемы в строке *s* и записывает нулевой символ (‘\0’) в *s* непосредственно за возвращенной лексемой.

```
#include <stdio.h>
#include <string.h>
#define DELIMITERS  " .,:;\n\t"      /* символы-разделители */
int main( )
{
  char sentence[500];     /* строка, разбиваемая на лексемы */
  char *word;       /* адрес начала очередной лексемы в строке */
  fgets(sentence, 500, stdin);
  word = strtok(sentence, DELIMITERS);
  while (word != NULL)
  {
    puts(word);
    word = strtok(NULL, DELIMITERS);
  }
  return 0;
}
```

При последующих обращениях с нулевым значением первого аргумента функция strtok() продолжает разбивать ту же строку на лексемы. Аргумент NULL показывает, что при вызове strtok() должна продолжать разбиение с того места строки sentence, которое было записано при последнем вызове функции.

Функции преобразования строк

Ниже приводятся функции преобразования строк (библиотека <stdlib.h>).

Прототип	Описание функции
double atof(const char *s)	переводит строку *s* в действительное число
int atoi(const char *s)	переводит строку *s* в целое число
long atol(const char *s)	переводит строку *s* в длинное целое число
double strtod(const char *s, char **endp)	преобразует первые символы строки *s* в тип double, игнорируя начальные символы-разделители; если endp не NULL, то указателю *endp присваивается адрес символа, который является первым символом строки-остатка после преобразования
long strtol(const char *s, char **endp, int base)	преобразует первые символы строки *s* в тип long, игнорируя начальные символы-разделители; указателю *endp присваивается адрес символа, который является первым символом строки-остатка после преобразования; base – основание, по которому производится преобразование
unsigned long strtoul(const char *s, char **endp, int base)	работает так же, как и strtol(), только возвращает результат типа unsigned long

Например, ниже приводятся две программы, выполняющие одно и то же действие за счет двух эквивалентных инструкций

x = atof(s); и x = strtod(s, NULL);

```
/* пример функции atof() */
#include<stdio.h>
#include<stdlib.h>
int main( )
{
  char *s = "1.5";
```

```
/* пример функции strtod() */
#include<stdio.h>
#include<stdlib.h>
int main( )
{
  char *s = "1.5";
```

```
  double x;
  x = atof(s);
  printf("x = %f\n", x);
  return 0;
}
```

```
  double x;
  x = strtod(s, NULL);
  printf("x = %f\n", x);
  return 0;
}
```

Если требуется произвести обратное преобразование числа в строку, тогда в этом поможет функция sprintf(). Например:

```
int a = 123;
char s[10];
sprintf(s, "%d", a);
puts(s);
```

Условие 1. Выделить и вывести на экран все слова произвольной строки. Слова отделяются друг от друга одним или несколькими пробелами.

```
char st[100], sl[100];
int k=0, i;
 gets(st);
 strcat(st," ");
  int n=strlen(st);
       if (n<2) return 1;
  sl[0]='\0';
  for (i=0; i<n; i++)
  if (st[i] != ' ')
  {
     sl[k]=st[i];
     sl[k+1]='\0';
     k++;
  }
  else
  {
    if (strlen(sl)>0) puts(sl);
         sl[0]='\0';
     k=0;
  }
```

Условие 2. Определить, является ли строка палиндромом, т.е. читается ли она слева направо так же, как и справа налево.

```
char st[80]="A roza   upala na  lapu Azora";
int i,j;
bool bl=true;
strlwr(st);
 i=0; j=strlen(st)-1;

 while (i<=j) {
   while (st[i]==' ') i++;
   while (st[j]==' ') j--;
if (st[i] !=st[j])
{
   bl=false;
      break;
}
         i++; j--;
 }
if (bl)  cout << "Palindrom" << endl;
   else cout << "Ne palindrom" << endl;
```

Задания для самостоятельной работы

Задание 1. Требуется выделить в строке-предложении s все слова, разделенные символами-разделителями «_.,;:\n\t!?» и обработать выделенные слова в соответствии с вариантом задания.

Определения

Регулярное слово – слово, состоящее только из больших латинских букв.

Палиндром – это слово, которое одинаково читается слева направо и справа налево.

Алфавитный порядок задается таблицей ASCII.

Варианты заданий

A1. Напечатать все слова, начинающиеся на большую и заканчивающиеся на меленькую букву.

A2. Напечатать все слова, содержащие хотя бы одну цифру.

A3. Напечатать все слова, содержащие хотя бы одну маленькую латинскую букву.

A4. Напечатать все слова, содержащие хотя бы одну большую латинскую букву.

A5. Напечатать все слова, состоящие только из маленьких латинских букв.

A6. Напечатать все слова, состоящие только из больших латинских букв.

A7. Напечатать все слова, состоящие только из цифр.

A8. Напечатать все слова, состоящие не менее чем из четырех букв.

A9. Напечатать все пятибуквенные слова.

A10. Напечатать все слова, содержащие буквы и цифры.

A11. Напечатать все слова, не содержащие ни одной цифры.

A12. Напечатать все слова, не содержащие ни одной большой латинской буквы.

A13. Напечатать все слова, содержащие хотя бы две цифры.

A14. Напечатать все слова, содержащие две рядом стоящие буквы.

A15. Напечатать все слова, содержащие две рядом стоящие цифры.

Задание 2.

1. Даны два слова (две переменные). Сколько раз во втором слове встречается первая буква первого слова.
2. Даны два текста (две переменные). Вычислить количество предложений в каждом из них.
3. Даны два слова по 8 символов (две переменные). Сколько раз во втором слове встречается последняя буква первого слова.
4. Дан текст. Определить в каких позициях в нем встречается символ «;» (другими словами : вывести на экран номера позиций в которых встречается символ «;»).
5. Дан текст. Переставить в нем первую букву первого слова и первую букву последнего слова . (Сначала найти номер последнего пробела).
6. Дан текст. Определить в каких позициях в нем начинается каждое новое предложение (сначала найти позиции точек).
7. Даны два слова (две переменные). Сколько раз во втором слове встречается третья буква первого слова.

8. Дан текст. Переставить в нем первую букву первого предложения и первую букву последнего предложения. (Сначала найти номер последней точки без учета точки в конце всего текста).

9. Даны два слова (две переменные). Сколько раз в первом слове встречается третья буква второго слова.

10. Дан текст. Переставить в нем первую букву первого предложения и первую букву второго предложения. (Сначала найти номер первой точки).

11. Дан текст. Сколько раз в нем встречается символ «=».

12. Дан текст. Определить в каких позициях в нем начинается каждое новое слово (сначала найти позиции пробелов).

13. Даны два текста (две переменные). В каком из них больше слов? При условии, что слова разделяются только одним пробелом. Сначала найти количество пробелов в каждом тексте.

14. Дан текст. Переставить в нем первую букву первого слова и первую букву второго слова. (Сначала найти номер первого пробела).

15. Даны два слова (две переменные). Сколько раз в первом слове встречается последняя буква второго слова.

Задание 3. Обработка цифр в строке.

№ варианта	Задание
1	Дан текст. Найти максимальное из имеющихся в нем чисел.
2	Дан текст. Найти сумму всех имеющихся в нем чисел.
3	Дан текст, в котором имеется несколько идущих подряд цифр. Получить число, образованное этими цифрами.
4	Дан текст. Найти наибольшее количество идущих подряд цифр.
5	Дан текст, имеющий вид: "$d_1 \pm d_2 \pm \ldots \pm d_n$", где d_i — цифры $(n > 1)$. Вычислить записанную в тексте алгебраическую сумму.
6	Дан текст, имеющий вид: "$d_1 - d_2 + d_3 - \ldots$", где d_i — цифры $(n > 1)$. Вычислить записанную в тексте алгебраическую сумму.
7	Дан текст, имеющий вид: "$d_1 + d_2 + \ldots + d_n$", где d_i — цифры $(n > 1)$. Вычислить записанную в тексте сумму.
8	Дан текст, представляющий собой десятичную запись целого числа. Вычислить сумму цифр этого числа.
9	Дан текст. Определить, является ли он правильной десятичной записью целого числа.
10	Дан текст, в начале которого имеются пробелы и в котором имеются цифры. Найти порядковый номер максимальной цифры, начиная счет с первого символа, не являющегося пробелом. Если максимальных цифр несколько, то должен быть найден номер первой из них.
11	Дан текст, в котором имеются цифры. а) Найти их сумму. б) Найти максимальную цифру.
12	Дан текст. Определить количество цифр в нем.
13	Дан текст. Напечатать все имеющиеся в нем цифры.
14	Дан символ. Выяснить, является ли он цифрой.
15	Дан текст, имеющий вид: "$d_1 + d_2 + \ldots + d_n$", где d_i — цифры $(n > 1)$. Вычислить записанную в тексте сумму.