Символьные данные и строки

Лекция 15а-15б

Методические указания к работе с лекцией.

1. Лекция состоит из трёх разделов.
2. Внимательно проработать слайды с 4 до 16, с описанием работы с символьными данными, изучив в деталях коды примеров.
3. На слайдах 17-19 итоги и контрольные вопросы по первой части. ***Написать ответы на эти вопросы*** и вставить их в ответное письмо мне.
4. Вторая часть лекции (слайды 20-25) – функции по работе со строками. ***Переписать, не скопировать***(!) все функции себе, чтобы они были под рукой во время выполнения ЛР и ***на экзамене (!)***
5. Слайды 26-33 : строки и указатели. Сложная тема – внимательно разобрать примеры!
6. ***Слайд 34 –Вопросы по второй части лекции. Написать ответы на них и вставить в ответное письмо.***
7. Слайды 35-36: примеры задач по работе со строками. **Внимательно проанализировать коды программ**!
8. Слайды 49-50 – В своих ***библиотеках сохранить программы*** перевода из кодировки DOS к кодировке Windows и обратно.
9. На слайде 57 представлено задание на самостоятельную работу. Решение этого задания прислать мне на e-mail: nk_petrova@mail.ru вместе с ответами на вопросы лекции.
10. ***Слайд 60: вопросы по третьей части лекции. Ответы выслать мне ответным письмом.***

Аннотация

В лекции рассматриваются

- понятия и определения символьных данных и строк,
- сходство и отличия их внутреннего представления,
- способы объявления, инициализация строк,
- методы доступа к элементам строк,
- определение размера строк,
- различные способы организации ввода/вывода символьных данных и строк.

Описание символьных данных

Для **символьных** данных в C++ введен тип char. Описание символьных переменных:

```
char список_имен_переменных;
```

Например:

```
const char c='c';
  //символ – занимает один байт, его значение не меняется
char a,b;
  /*символьные переменные, занимают по одному байту,
    значения меняются*/
const char *s="Пример строки\n";
  //текстовая константа
```

Описание строковых данных

Строка – это последовательность символов, заключенная в двойные кавычки (" ").

Размещая строку в памяти, транслятор автоматически добавляет в ее конце символ '\0' (нулевой символ или нулевой байт, который является признаком конца строки). В записи строки может быть и один символ: "A" (заключен в двойные кавычки), однако, в отличие от символьной константы 'A' (используются апострофы), длина строки "A" равна 2 байтам.

В языке C++ строка – это пронумерованная последовательность символов (массив символов), она всегда имеет тип char[]. Все символы строки нумеруются, начиная с нуля. Символ конца строки также нумеруется – ему соответствует наибольший из номеров. Таким образом, строка считывается значением типа "массив символов". Количество элементов в таком массиве на 1 больше, чем изображение соответствующей строки, так как в конец строки добавлен нулевой символ '\0'

Особенности работы со строковыми данными

Символьная строка в программном коде может располагаться на нескольких строках. Для переноса используется символ '\' с последующим нажатием клавиши ввод. Символ '\' игнорируется компилятором, и следующая строка считается продолжением предыдущей.

Присвоить значение строке с помощью оператора присваивания нельзя, так как для массивов не определена операция прямого присваивания. Поместить строку в символьный массив можно либо при вводе, либо с помощью инициализации:

```
char s1[] = "ABCDEF";     //инициализация строки
```

```
char s2[]={'A','B','C','D','E','F','\0'};    //инициализация строки
```

Операция вычисления размера (в байтах) sizeof действует для объектов символьного типа и строк.

Пример 1. Расчёт памяти символьных данных

```
char s1[10]="string1";
  int k=sizeof(s1);
  cout<<s1<<"\t"<<k<<"\n";
char s2[]="string2";
  k=sizeof(s2);
  cout<<s2<<"\t"<<k<<"\n";
char s3[]={'s','t','r','i','n','g','3','\0'};
  /*окончание строки '\0' следует соблюдать, формируя
    в программах строки из отдельных символов*/
  k=sizeof(s3);
  cout<<s3<<"\t"<<k<<"\n";
const char *s4="string4";
  //указатель на строку, ее нельзя изменить
  k=sizeof(s4);
  cout<<s4<<"\t"<<k<<"\n";
```

ОБРАТИМ ВНИМАНИЕ НА 4-ю строку результатов: переменная k вернула размер не массива символов, сопоставленных с указателем *s4, а размер в байтах самого указателя,
а указатели любого типа (ПОМНИМ!) занимают в ОП 4 байта.

```
string1 10
string2 8
string3 8
string4 4
```

1) Ввод-вывод одиночного символа

getchar() – функция (без параметров) используется для ввода одиночного символа из входного потока. Она возвращает 1 байт информации (символ) в виде значения типа int. Это сделано для распознавания ситуации, когда при чтении будет достигнут конец файла.

putchar(ch) – функция используется для вывода одиночного символа, то есть помещает в стандартный выходной поток символ ch. Аргументом функции вывода может быть одиночный символ (включая знаки, представляемые управляющими последовательностями), переменная или функция, значением которой является одиночный символ.

Пример 2. Введите предложение, в конце которого стоит точка, и подсчитайте общее количество символов, отличных от пробела (не считая точки)

```
…
char z; //z - вводимый символ
  int k; //k - количество значащих символов
  printf("Напишите предложение с точкой в конце:\n");
  for (k=0; (z=getchar())!='.';)
  /*выражение z=getchar() заключено в скобки, так как
    операция присваивания имеет более низкий приоритет,
    чем операция сравнения*/
    if (z!=' ')
      k++;
  printf("\nКоличество символов=%d",k)
…
```

Результат выполнения программы:

```
Напишите предложение с точкой в конце:
1 2 3 4 5 6 7 8 9 0.
Количество символов=10
```

2) Ввод-вывод стандартного текстового (символьного) потока

gets(s) – функция, которая считывает строку s из стандартного потока до появления символа '\n', сам символ '\n' в строку не заносится.

puts(s) – функция, которая записывает строку в стандартный поток, добавляя в конец строки символ '\n', в случае удачного завершения возвращает значение больше или равное 0 и отрицательное значение (EOF = -1) в случае ошибки.

Для Visual Studio 2015 и выше эти функции устарели. Безопасные версии этих функций, gets_s и _getws_s, по-прежнему доступны.

Пример 3. Вычислите длину введенной строки.

Подобного рода примеры будут включены в задачи на экзамене

```
…
char st[100];
  int i=0;
  puts("Введите строку:");
  gets(st);
  while(st[i++]);
  printf("Длина введенной строки = %i\n",i-1);
…
```

3) Форматированный ввод-вывод символьных данных и строк

printf() – функция, осуществляющая форматированный вывод данных.

scanf() – функция, осуществляющая форматированный ввод данных.

%c – спецификатор формата ввода-вывода одиночного символа.

%s – спецификатор формата ввода-вывода строки символов.

Пример 4. Записать введенную строку символов в обратном порядке.

```
…
char st[80];
  char temp;
  int i,len=0;
  printf("\nВведите строку > ");
  scanf_s("%s", st);
  while (st[len++]); //вычисление длины строки
  len-=2; //поправка на символ конца строки и нумерацию с нуля
  for(i=0;i<len;i++,len--){
    temp=st[i];  //обмен символов
    st[i]=st[len];
    st[len]=temp;
  }
  printf("\nПолученная строка  > %s",st);
…
```

Результат выполнения программы: при вводе строки "123 456 789", чтение данных осуществляется побайтно до первого пробела, то есть в строку s занесется только первое слово строки "123\0".

Так как s – имя символьного массива, то есть адрес его начального элемента, операция & в функции scanf_s для строк не используется.

Некоторые особенности символьного типа char

1. Переменную типа char можно рассматривать двояко:
   - как целое число, занимающее 1 байт и способное принимать значения:
   - от 0 до 255 (тип unsigned char) или
   - от -128 до 127 (тип signed char)
   - и как один текстовый символ.
2. Сам же тип char может оказаться как знаковым, так и беззнаковым, в зависимости от операционной системы и компилятора.
3. Поэтому использовать тип char не рекомендуется, лучше явно указывать будет ли он знаковым (signed) или беззнаковым (unsigned).
4. Как и целые числа, данные типа char можно складывать, вычитать, умножать, делить, а можно выводить на экран в виде одного символа. Именно это и происходит при выводе символа через объект cout.
5. Если же нужно вывести числовое значение символа (также называемое ASCII-кодом), то значение символа необходимо преобразовать к типу int.

Пример 5.

```
#include<iostream>
   using namespace std;
   int main()
   {  unsigned char c='A'; // Константы char заключаются
                           // в одинарные кавычки
      cout<<c<<" "<<(int)c<<endl;
      c=126;     // char можно присвоить и числовое значен
ие
      cout<<c<<" "<<(int)c<<endl;
      return 0;    }
```

```
A 65
~ 126
Для продолжения нажмите любую клавишу
```

Знак ~ имеет код 126

Пример 6. Организация последовательного посимвольного считывания всего входного потока

```
#include<iostream>
using namespace std;
   int main()
   {
      unsigned char c;
      while(cin>>c) // Цикл пока считывание успешно
      {  // Делаем необходимые действия
      }
      return 0;
   }
```

Обратите внимание: оператор cin определяет условие работы цикла while

Ключевые термины

- **Внутренние коды символов** – целые числа, однозначно соответствующие символам во внутреннем представлении.
- **Инициализация строки** – *определение* значения строки.
- **Размер строки** – объем памяти, занимаемой строкой, выраженный в байтах.
- **Символ конца строки** – нулевой *байт*, являющийся признаком конца строки символов.
- **Символьная константа** – константа типа char.
- **Символьная переменная** – *переменная* типа char.
- **Строка** – это пронумерованная последовательность символов, заключенная в двойные кавычки.

Краткие итоги

a) Для представления текстовой информации в C++ используются символьные данные и строки.

b) В C++ не определен *строковый тип* данных, и строка представляется как *массив символов*.

c) Инициализировать строку можно как *массив символов*.

d) Признаком конца строки является нулевой символ.

e) Обратиться к элементу строки можно по индексу, который соответствует порядковому номеру элемента.

f) Нумерация элементов строки начинается с нуля. Размер строки определяется количеством входящих в нее символов.

g) В C++ предусмотрены различные способы ввода и вывода одиночных символов и строк: с помощью стандартных функций, с помощью потокового или форматированного ввода/вывода. При считывании строки с клавиатуры признак конца строки добавляется автоматически.

h) Каждому символу однозначно соответствует его внутренний код.

Вопросы для самопроверки

1. Почему в C++ не выполняется операция прямого присваивания значения строке?
2. Почему символ и строка, состоящая из одного символа, занимают разный объем памяти?
3. Почему в функции scanf_c("%s",string); не указывается обращение к переменной по адресу?
4. Допустима ли операция сравнения над символами? Если да, то каким образом определены отношения "больше" и "меньше"?
5. Какая из функций, gets или puts, заносит в поток управляющий символ '\n' и с какой целью?
6. Можно ли выполнить присваивание символьной переменной числового значения? Почему?
7. В чем различия результатов вывода символьной переменной со спецификаторами %d и %c?

Функции работы с символами и со строками

Особенности библиотечных функций работы со строками

При использовании библиотечных функций следует учитывать некоторые особенности их выполнения и представления символьных данных в памяти.

- Функции, работающие с регистрами, распространяются только на латиницу.
- В C++ некоторые параметры функций обработки символов принадлежат типу int (unsigned), поэтому, если число станет больше 128 (255), функция будет работать некорректно.
- Перед первым обращением к строке она должна быть объявлена и проинициализирована. Во многих случаях в качестве начального значения строки необходимо бывает задать пустую строку. Такую инициализацию можно выполнить с помощью вызова функции strcpy(s, "");, но более эффективным будет присваивание *s=0;. Кроме того пустую строку можно инициализировать char s[10]=""; или char s[10]="\0";, но при этом размер строки должен быть задан.
- Функции копирования (кроме strncpy) не проверяют длину строки. Размер строки-приемника должен быть больше, чем размер источника на 1 символ (для символа '\0').

Некоторые часто используемые стандартные функции

Все строковые функции используют заголовок

```
#include <cstring>
```

1. strcpy(**куда**, **откуда**)
   копирует содержимое строки **откуда** в строку **куда**.
2. strcat(**s1**, **s2**)
   к строке **s1** присоединяет строку **s2**, при этом строка **s2** остается без изменений.
3. strcmp(**s1**, **s2**)
   сравнивает две строки и возвращает 0, если они равны. Если **s1** больше **s2** лексикографически (т.е. по порядку следования символов алфавита), возвращается положительное число, если она меньше **s2**, то возвращается отрицательное число.
4. strlen(**s**)
   возвращает длину строки **s**.

5. ispunct(**символ**)

если **символ** является знаком пунктуации, то функция возвращает ненулевое значение, в противном случае возвращает нуль.

К знакам пунктуации относятся все печатаемые символы,
не являющиеся буквами, цифрами и пробелами.

6. isspace(**символ**)

если символ является
- пробелом,
- знаком табуляции,
- символом возврата каретки или
- перехода на новую строку,

то функция возвращает **ненулевое значение**, **иначе** – возвращает **нуль**.

7. strstr(**строка1**, **строка2**)

выполняет поиск подстроки **строка1** в строке **строка2**.

Обе строки должны завершаться нуль-символами.

В случае успешного поиска функция возвращает **указатель** на найденную подстроку, в случае неудачи – NULL.

8. strtok(**строка**, **строка_разделителей**)

выполняет поиск следующего **слова** в строке **строка**.

Строка_разделителей состоит из символов, являющихся **разделителями**, определяющими **слово**.

В случае успешного поиска функция возвращает **указатель** на начало найденного слова, в случае неудачи – NULL.

Для определения начала слова функция сначала определяет символы, не содержащиеся в строке **строка_разделителей**. А затем посимвольно проверяет остальную часть строки до первого символа-разделителя, который сигнализирует конец слова.

Этот конечный маркер автоматически заменяется нулевым символом, и слово возвращается функцией. После этого, следующие вызовы функции strtok начинаются с этого нулевого символа.

Функции для преобразования типов

Функция	Прототип	Краткое описание действий
atof	`double atof (const char *str);`	преобразует строку `str` в *вещественное число* типа `double`
atoi	`int atoi (const char *str);`	преобразует строку `str` в целое число типа `int`
atol	`long atol (const char *str);`	преобразует строку `str` в целое число типа `long`
itoa	`char *itoa (int v, char *str, int baz);`	преобразует целое `v` в строку `str`. При изображении числа используется основание `baz` `(2<=baz<=36)`. Для отрицательного числа и `baz` =10 первый символ "минус" (–).
ltoa	`char *ltoa (long v, char *str, int baz);`	преобразует длинное целое `v` в строку `str`. При изображении числа используется основание `baz (2<=baz<=36)`.
ultoa	`char *ultoa (unsigned long v, char *str, int baz);`	преобразует беззнаковое длинное целое `v` в строку `str`

Строки и указатели

Строки в языке C++ представляют собой массив символов. Поскольку имя массива без индексов является указателем на первый элемент этого массива, то *при использовании функций обработки строк им будут передаваться не сами строки, а указатели на них.*

Так как все строки в языке C++ заканчиваются нулевым символом, который имеет значение <ложь>, то условие в операторе while(*str) будет истинным до тех пор, пока программа не достигнет конца строки.

Пример 7: для работы со строками в большинстве случаев целесообразно применять указатели.

При разработке функций Приведем примеры фрагментов программ :

```
/*Пример пользовательской функции
char * strcpy_my (char *s1, char *s2){
  char *ptrs1 = s1;
  //указатель инициализирован на нача
  while ((*s1++ = *s2++) != 0);
  return ptrs1; //возвращается указатель
}
```

```
                                012345\0        0123\0
char s1[10] = "ABCDEF", s2[10]="1234";
char *ptr;
cout << s1 << "\t" << s2 << '\t' <<  "\n";
ptr= strcpy_my(s1, s2);
cout << s1 << "\t" << s2 << '\t' <<  "\n";
```

```
ABCDEF  1234
inside strcpy_my: BCDEF 234
inside strcpy_my: CDEF  34
inside strcpy_my: DEF   4
inside strcpy_my: EF
1234    1234
```

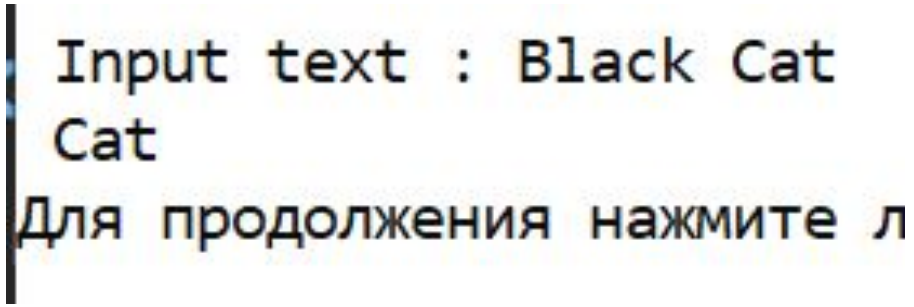
Пример 8: использование нулевого ограничителя упрощает различные *операции* над строками.

```
/*Пример пользовательской функции конкатенации*/
char * strcat_my (char *s1, char *s2) {
char *p1, *p2;
p1 = s1; p2 = s2;
while ( *p1 != '\0') p1++; //найти конец 1-ой строки.
                    //или while ( *p1) p1++;
  while ((*p1 = *p2) != 0) {
  /*копировать строку p2, пока не будет скопирован нулевой     ограничитель*/
    p1++;
    p2++; //Передвинуть указатели к следующему байту
  }      //или while (( *p1++ = *p2++) != 0);/*.
return s1;
}
```

Пример 9: выделение подстроки с помощью «&»

Можно присваивать указателю адрес любого отдельного элемента массива путем добавления символа '&' к индексированному имени. Программа выводит на экран часть введенной строки после первого пробела:

```
int main() {
    char s[80], *p;
    int i;
     cout<<“ Input text: ";
    gets_s(s);
    /*найти первый пробел или конец строки*/
    for (i = 0;  s[i] && s[i] != ' ‘;  i++);
    p = &s[i];
    cout <<p<< endl;
    system("pause");
    return 0;
}
```



Особенности передачи строк в функции

При использовании строк или указателей на строки в качестве параметров функций следует учитывать некоторые особенности.

1. При передаче строки как параметра функции ***не указывается длина***, так как ограничителем является символ конца строки.

2. Строки передаются в функции в качестве параметров как массивы символов или как указатели типа char.

3. При побайтовом копировании строки или ее подстроки без использования стандартных функций ***формируемую строку следует завершить, дописав символ конца строки.*** В противном случае строка не воспринимается как единое целое, а при выходе за ее границы доступными становятся байты, содержащие "мусор", то есть непредсказуемую информацию.

4. Обращение к строкам через указатели позволяет вносить и сохранять изменения, записанные в адресуемой области памяти. Для недопущения изменений в строке указатель на константу можно объявить с лексемой **const** следующим образом: **const char *p;.**

5. В силу специфики ***представления строк в виде символьного массива*** сами строки, строковые константы, заключенные в кавычки, и указатели на строки обрабатываются эквивалентно. При этом каждый такой элемент адресует область памяти и ***передается в функции как адрес.***

6. При копировании строки или подстроки с использованием указателя ***не создается физической копии значений элементов.*** Объявленный новый указатель адресует то место в памяти, с которого начинается копируемая строка или подстрока. Например:

```
char text[50]="Язык программирования";
char *p=text, *pp;
//объявление и инициализация указателя p адресом строки text
pp=p;
//указатель pp адресует ту же строку text
```

Адресация на тот же участок памяти объясняется, во-первых, неэффективностью повторного хранения уже имеющихся данных, во-вторых, относительной программной трудоемкостью копирования байтов, в-третьих, для хранения адреса строки требуется гораздо меньше места, чем для самой строки.

Ключевые термины

Адрес строки – это указатель на блок непрерывной области памяти, с которого начинает располагаться массив символов.

Строки как параметры функций – это описание передачи значений строк в функции как массив символов или указатель типа char.

Указатель на строку – адрес начала расположения строки в памяти.

Начиная с Visual Studio 2015 многие стандартные функции устарели и на замену им можно использовать альтернативные функции, например: strcpy strcpy_s; strcat strcat_s; scanf scanf_s;

Краткие итоги

1. В силу специфики представления строк в виде символьного массива сами строки, строковые константы, заключенные в кавычки, и указатели на строки обрабатываются эквивалентно.
2. Строки передаются в функции в качестве параметров как массивы символов или как указатели типа char.
3. Обращение к конкретному элементу строки можно осуществить посредством адресации индексированного имени строки.
4. При формировании строки без использования стандартных функций требуется дописывать символ конца строки.
5. С помощью указателей на константы можно защитить строку от изменений.
6. Копирование строк с помощью указателей осуществляется через объявление нового указателя, адресующего область памяти, занимаемую строкой или подстрокой.

Контрольные вопросы

1. Почему обращения к строке через ее имя и через указатель эквивалентны?
2. Почему в качестве параметра функции передается адрес строки, а не сама строка символов?
3. Возможно ли применение операций инкремента и декремента к указателю на строку? Если да, то что будет адресовать полученный указатель?
4. Почему при формировании строки без использования стандартных функций необходимо дописывать символ конца строки? Почему этого не требуется при считывании строк с клавиатуры?
5. Какие возможны ошибки в программе при некорректной работе со строками?
6. Для защиты строки от изменения объявляется указатель на константу или указатель-константа? Почему?

Примеры программ по операциям со строками

Задача 1. Поиск подстроки

Составить программу, которая определяет, встречается ли в заданном **текстовом файле**
искомая **последовательность символов**.

Текстовый файл содержит несколько строк.

Длина каждой строки не превышает 80 символов.

Текст не содержит переносов слов.

Последовательность не содержит пробелов.

Словесный алгоритм программы

1. Поскольку переносы отсутствуют, можно ограничиться поиском заданной последовательности в каждой строке отдельно.
2. Следовательно, необходимо помнить только одну текущую строку файла.
3. Для ее хранения выделим символьный массив длиной 81 символ (с учетом нуль-символа).
4. Искомая последовательность символов вводится с клавиатуры. Ее длина не более 80 символов.
5. Для ее хранения также выделим символьный массив длиной 81 символ (с учетом нуль-символа).
6. Результатом работы программы является сообщение либо о наличии заданной последовательности, либо об ее отсутствии.

```
#include <iostream>
#include <fstream>     // для работы с файлами
#include <cstring>     // для работы с символьными массивами

using namespace std;
int main( )
{
   const int len = 81;     // длина последовательности
   char word[len];      // последовательность символов
   char line[len];     // строка из файла
   cout << "Enter word: ";   // приглашение к вводу
   cin >> word;    // ввод искомой послед-ти символов
    ifstream fin("TextFile1.txt");     // открытие файла для чтения
    if(!fin)     // проверка НЕуспешности открытия файла
        cout << "Error file";  // вывод сообщения
```

```
else
{     // файл успешно открыт
    while(!fin.eof( ) )    // цикл пока не достигнут конец файла
    {
       fin.getline(line, len);   // чтение строки из файла

        if(strstr(line, word))    // поиск последовательности в строке
        {    // посл-ть в строке есть
         cout << "Yes word " << endl;
         return 0;       // выход из программы
         }
     } // конец цикла чтения строк из файла

      cout << "No word" << endl;     // посл-ть отсутствует
  } // конец блока else
  fin.close( );  // закрытие файла
  return 0;
}   // конец main( )
```

Задача 2. Подсчет количества вхождений слова в текст

Составить программу, которая определяет, сколько раз встретилось заданное слово в текстовом файле.

Текстовый файл содержит несколько строк.

Длина каждой строки не превышает 80 символов.

Текст не содержит переносов слов.

Заданное слово не содержит пробелов.

```
#include <iostream>
#include <fstream>
using namespace std;
int main( )
{
    const int len = 81;  // длина слова
    char word[len];  // слово
    char line[len];  // строка из файла
    char delims[ ] = " .,!&//<>|)(*:;\"";  // массив разделителей
    cout << "Enter word: " ;  // приглашение к вводу слова
    cin >> word;          // ввод искомого слова
```

```
ifstream  fin("TextFile1.txt");    // открытие файла для чтения
if(!fin)      // проверка успешности открытия файла
{
   cout << "Error  file" << endl;
   return 1;
}
else
{
   char *token;    // указатель на текущее слово (token - символ)
   int count=0;    // кол-во повторений слова
```

```
while(fin.getline(line,  len))  // чтение очередной строки из файла –
{  // начало цикла чтения строк
  // если уже конец файла, то NULL – выход из цикла
  token = strtok(line,  delims);  // при первом вызове формирует
                    // адрес первого слова в строке line
  while(token != NULL)    // цикл до окончания строки
  {
    if(!strcmp(token,  word))  // сравнение слов
      count++;  //очередное слово совпало с заданным - кол-во слов + 1
        // здесь можно сравнивать,
       //  т.к. ф-я strtok заменяет на NULL разделитель после слова

    token = strtok(NULL,  delims);   // поиск следующего слова
  }

 }
```

Внутренние коды символов

В языке C++, как уже было рассмотрено ранее, принято соглашение, что везде, где *синтаксис* позволяет использовать целые числа, можно использовать и символы, то есть данные *типа char*, которые при этом представляются числовыми значениями своих внутренних кодов.

Такое соглашение позволяет сравнительно просто упорядочивать символы, обращаясь с ними как с целочисленными величинами. Например, внутренние коды десятичных цифр в таблицах кодов *ASCII* упорядочены по числовым значениям, поэтому несложно перебрать символы десятичных цифр в нужном порядке.

Однако, при работе с буквами кириллицы возникает серьёзная проблема при вводе выводе

Почему существует две кодировки для символов?

Проблема заключается в том, что когда в Microsoft придумывали Windows, то попутно *придумали новую кодировку для кириллицы*. Трудно сказать зачем, но придумали. А старую кодировку, которая использовалась в MS DOS, оставили. Видимо в целях обратной совместимости.

С выходом новых версий Windows ситуация ухудшилась. Т.к. *консоль,* уже как часть операционной системы, *унаследовала кодировку кириллицы от MS DOS*.

В итоге сейчас для кириллицы имеем две кодировки: **cp866** — старая досовская кодировка и **cp1251** (она же windows-1251) — новая, от Windows.

В настоящее время дело осложняется тем, что окончательно созрел Unicode, что дает еще несколько кодировок не совместимых с cp1251 и с cp866, и не совсем совместимых между собой.

«cp» в названии кодировки означает codepage — «страница кодировки символов»

Проблемы использования двух кодировок

Консоль имеет собственную настройку кодовой страницы. Для России по умолчанию это — cp866. Настройка кодовой страницы консоли может быть изменена командой chcp <номер кодовой страницы>.

Итак, при написании программ строки могут встречаться в двух различных кодировках в следующих местах:

В исходных текстах программ в виде литералов.

Вывод на консоль.

Ввод с консоли.

Вывод в файл.

Процессор работает с кодировкой cp1251

Коды символов типа CHAR

Символ	Код MS DOS (CP 866)		Код Windows (CP 1251)		DOS→Win
	unsigned char от 0 до 255	char от -128 до +127	unsigned char от 0 до 255	char от -128 до +127	Δ
А…Я	128…159	- 128… - 97	192…223	- 64…- 33	+64
а...п	160…175	- 96…- 81	224…239	- 32…- 17	+64
р…я	224…239	- 32…- 17	240…255	- 16…- 1	+16
ё	241	- 15	184	- 72	замена кода
Ё	240	- 16	168	- 88	замена кода

2. Изменение кодировки символа

Составить программу с функциями
для преобразования кода символа
из ср866 (MS DOS) в код ср1251 (Windows).

Программа должна выполнять следующие задачи:

1. Ввод символа с клавиатуры (ср866).
2. Запись этого символа в текстовый файл (ср1251).
3. Чтение символа из текстового файла (ср1251).
4. Вывод символа на экран (ср866).

Начало программы.

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
//функция для преобразования символа
// из кода ср866 в ср1251

void DosToWin(char d, char &w)
{
    if (( d >= -128) && ( d <= -81) )   // А...Я  а...п
       w = d + 64;
    if (( d >= -32) && ( d <= -17))     // р...я
       w = d + 16;
    if (d == -16)   w = -88;     // Ё
    if ( d == -15)  w = -72;     // ё
}
```

Символ	СР 866	СР 1251	DOS→Win
А…Я	- 128… - 97	- 64…- 33	+64
а...п	- 96…- 81	- 32…- 17	+64
р…я	- 32…- 17	- 16…- 1	+16
ё	- 15	- 72	замена
Ё	- 16	- 88	замена

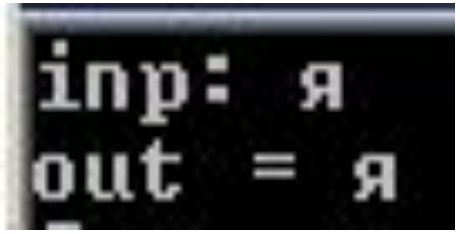
```
//функция для преобразования символа
// из кода  cp1251 в cp866

void WinToDos(char w, char &d)
{
    if (( w >= -64) && ( w <= -17) )   // А...Я  а...п
       d = w - 64;
    if (( w >= -16) && ( w <= -1))     // р...я
       d = w - 16;
    if (w == -88)   d = -16;     // Ё
    if ( w == -72)  d = -15;      // ё
}
```

Символ	CP 866	CP 1251	Win→DOS
А…Я	- 128… - 97	- 64…- 33	-64
а...п	- 96…- 81	- 32…- 17	-64
р…я	- 32…- 17	- 16…- 1	-16
ё	- 15	- 72	замена
Ё	- 16	- 88	замена

```
int main( )
{
   char inp,  // входная переменная
        out;  // выходная переменная
   ofstream fout("out.txt");      // создание и открытие
                          //   файла для записи
   cout << "inp: ";       // приглашение к вводу
   cin >> inp;          // ввод значения
                      //  входной переменной
   DosToWin(inp, inp);  // обращение
                      //  к функции преобразования кода
   fout << inp;  // запись в файл
   fout.close(); // закрытие файла
```

```
 ifstream finp( "out.txt" );  // открытие файла для чтения
 finp  >>  out;      // чтение из файла символа
                  // в выходную переменную
 finp.close( );     // закрытие файла
 WinToDos(out, out);     // обращение к функции
                        //  преобразования кода
 cout << "out = " << out << endl;  // вывод на экран
                           //   значения выходной переменной
 return 0;
}
```

Результат работы программы:



3. Вставка символа в строку

Составить программу с функцией для вставки символа в строку.

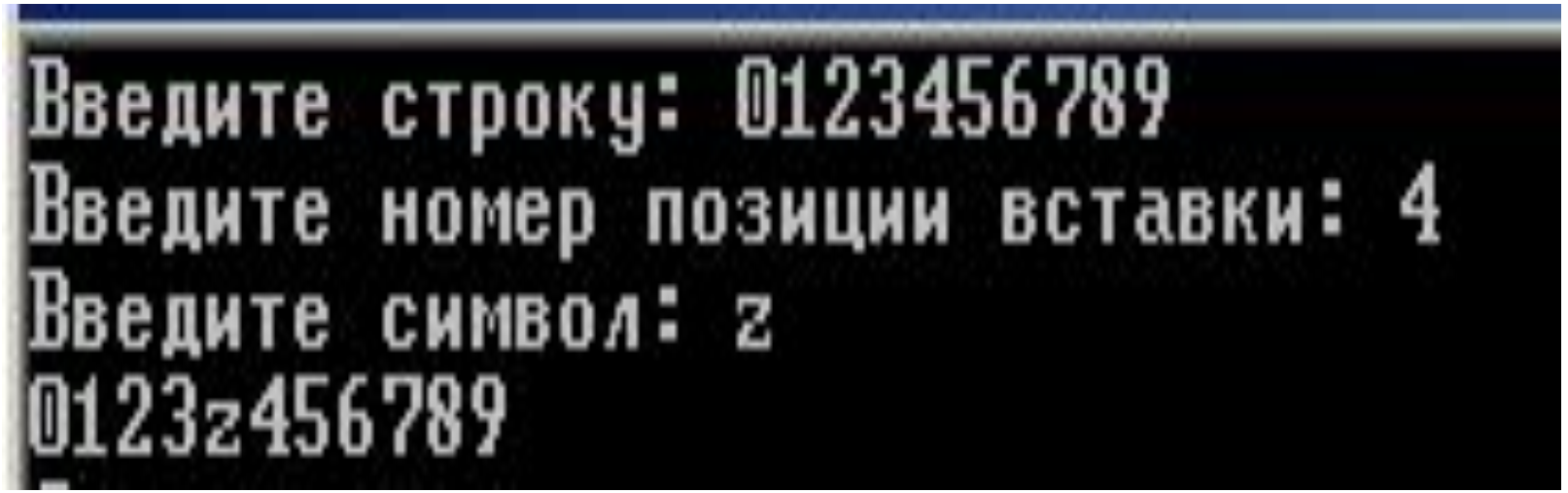
Параметры функции:
строка, позиция вставки, вставляемый символ.

Использовать эту функцию в основной функции, где исходная строка, номер позиции и вставляемый символ задаются при вводе.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

// функция вставки символа
// str[  ]  - входная строка
//  p – позиция вставки (входная величина)
//  c – вставляемый символ (входная величина)
void insert(char str[  ], int p, char c)
{    // цикл от конца строки до места вставки
   for( int  i = strlen(str);    i >= p;    i-- )
       str[ i + 1 ] = str[ i ];   // символы сдвигаются к концу строки
   str[p] = c;   // запись символа в освободившеюся позицию
}
```

```
int main( )
{
        setlocale(0, "");           // поддержка кириллицы
        const   int   L = 20;       // длина  строки
        char   c;          //  введенный символ
        char   s[L];     // строка
        int    n;      // номер позиции вставки
        cout   <<  "Введите строку: ";
        cin.getline(s,  L);  // считывание строки с клавиатуры
        cout  <<  "Введите номер позиции вставки: ";
        cin  >>  n;   // ввод номера позиции вставки
        cout   <<   "Введите символ: ";
        cin  >>  c;  // ввод вставляемого символа
        insert( s, n, c);  // обращение к функции
        cout  <<  s  <<  endl;  // вывод результата на экран
        return 0;
}
```

Результат работы программы



Задание на самостоятельную работу

Доработать данную программу:

- добавить функцию **удаления** символа из указанной позиции
- ввести **диалог**: удалить или вставить
- обеспечить вставку символов **русского алфавита**, а также символов латинского алфавита, цифр и т.д.

Ключевые термины

- **Конкатенация строк** – это результат последовательного *соединения строк*.
- **Лексикографический порядок** – правило сравнения символов, основанное на величине кода внутреннего представления каждого символа.
- **Пустая строка** – это строка единичной длины, содержащая только *символ конца строки*.
- **Сравнение строк** – это результат проверки выполнения отношения "больше", "меньше" или "равно" над строками.
- **Стандартные функции по работе со строками** – это функции обработки строк, прототипы которых входят в *стандартные библиотеки* C++.

Краткие итоги

a) Для работы со строками в языке C++ предусмотрены стандартные функции, прототипы которых включены в *стандартные библиотеки* stdlib.h и string.h.

b) При *обращении к функциям* для работы со строками следует учитывать, что изменение значений элементов строк сохраняются после завершения работы функции.

c) Перед использованием строки в программном коде ее необходимо проинициализировать. Неинициализированные строки могут привести к некорректной работе программы.

d) В некоторых стандартных функциях по работе со строками следует проводить контроль длин параметров.

e) Результат работы некоторых функций требует принудительного добавления к строке *символа конца строки*.

f) Значения элементов строк зависят от регистра.

g) Изменение регистра символов кириллицы в программе может выполняться некорректно.

Вопросы для самопроверки

1. Что будет являться результатом работы функции побайтового копирования строк, если *длина строки*-источника превосходит допустимый размер строки-приемника?
2. Что будет являться результатом работы функции побайтового копирования строк, если *длина строки*-источника меньше размера строки-приемника?
3. Почему при сравнении строк важен регистр символов?
4. Как сравниваются строки разной длины?
5. Какие возможны последствия при обращении к неинициализированной строке?