Лабораторная работа №1. Работа со строками в С++.

1 Цель и порядок работы

Цель работы –освоить основные операции работы со строками.

Порядок выполнения работы:
- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя, согласно своему варианту;
- написать программу и отладить ее на ЭВМ.

2 Краткая теория. Строки в С++.

2.1 Ввод-вывод строк

В С++ есть два вида строк С-строки и класс `string` стандартной библиотеки С++.

С-строка представляет собой массив символов, завершающийся символом с кодом 0. Класс `string` более безопасен в использовании, чем С-строки, но и более ресурсоемок. Для грамотного использования этого класса требуется знание объектно-ориентированного программирования.

Память под строки, как и под другие массивы, может выделяться как компилятором, так и непосредственно в программе. Длина динамической строки может задаваться выражением, длина нединамической 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 = 100;
char msg[len_str];
```

При задании длины необходимо учитывать завершающий нуль-символ. Например, в строке, приведенной выше, можно хранить не 100 символов, а только 99. Строки можно при описании инициализировать строковыми константами, при этом нуль-символ в позиции, следующей за последним заданным символом, формируется автоматически:

```
const int len_str = 100;
char msg[len_str] = "Новая строка";
```

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

```
char msg[ ] = "Новая строка"; //13 символов
```

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[len_str];
```

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.

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 "stdafx.h"
#include <iostream>

using namespace std;

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

Строки вводится точно так же, как и переменные других типов.

При вводе строки из нескольких слов, программа выведет только первое слово. Это связано с тем, что ввод выполняется до первого пробельного символа (то есть пробела, знака табуляции или символа перевода строки '\n')

Если требуется ввести строку, состоящую из нескольких слов, в одну строковую переменную, используются методы `getline` или `get` класса `istream`, объектом которого является `cin`.

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main()
{
    const int n = 80;
    char s[n];
    cin.getline(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 также считывается (удаляется) из входного потока, но не записывается в 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 ноль.

Если в программе требуется ввести несколько строк, метод `getl1ne` удобно использовать в заголовке цикла, например:

```
#include "stdafx.h"
```

```
#include <iostream>

using namespace std;

int main()
{
    const int n = 80;
    char s[n];
    while (cin.getline(s, n))
    {
        cout << s << endl;
    };
    return 0;
}
```

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

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

Строка при этом выравнивается по правому краю отведенного поля. Если заданное количество позиций недостаточно для размещения строки, оно игнорируется, и строка выводится целиком.

Библиотека содержит также функции, специально предназначенные для ввода-вывода строк: `gets` и `puts`.

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 `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`.

2.2 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`, а для определения длинны строки – `strlen`.

```
#include "stdafx.h"
#include <iostream>
#include <string.h>


using namespace std;

int main()
{
   char a[100] = "Working with a strings";

   size_t m = strlen(a) + 1; //добавим 1 для учета нуль-символа

   char *p = new char [m];

   strcpy(p, a);
   strncpy(p, a, strlen(a) + 1);


   return 0;
}
```

Замечание. Использование функций `strcpy` и `strncpy` может быть небезопасным, так как они не проверяют размер буфера-приемника, что может привести к выходу за границы и затиранию чужих областей памяти. Выход за границы строки и отсутствие нуль-символа являются распространенными причинами ошибок в программах обработки строк. Для решения этой проблемы можно использовать безопасные версии функций: `strcpy_s` и `strncpy_s`, и избавить себя от собственноручного отслеживания размеров строки. При запуске программы компилятор выдает соответствующее предупреждение, которое можно проигнорировать в данном случае.

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

Функция `strcpy(dst, src)` копирует все символы строки, указанной вторым параметром (src), включая завершающий 0, в строку, указанную первым параметром (dst). Функция `strncpy(dst, src, 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 dst заполняются нуль-символами. В противном случае (если n меньше или равно длине строки src) завершающий нуль-символ в dst 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(src)` 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.

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
#include "stdafx.h"
#include <iostream>
#include <string.h>

using namespace std;

int main()
{
   char a[] = "15) Кол-во - 249 шт. Цена - 499.99 руб.";

   int num;
   long quantity;
   double price;

   num = atoi(a);
   quantity = atol(&a[12]);//12 - смещение начала кол-ва
   price = atof(&a[27]);   //27 - смещение начала цены

   cout << num << ' ' << quantity << ' ' << price;

   return 0;
}
```

Замечание. При переводе вещественных чисел разделитель целой и дробной части зависит от настроек локализации. По умолчанию используется символ точка. При изменении локализации (функция `setlocale(LC_ALL, "Russian")`), разделитель меняется на принятый в России, т.е. символ запятая.

Библиотека предоставляет также различные функции для, сравнения строк и подстрок, объединения строк, поиска в строке символа и подстроки и выделения из строки лексем.

2.3 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)`.

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()` и `get(char)`.

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.

Рассмотрим пример использования функций работы с символами.

```
//Пример применения функций работы со строками
#include "stdafx.h"
#include <iostream>
#include <stdio.h>

using namespace std;

int main()
{
   setlocale(LC_ALL, "Russian");

   char a, b, c, d, e;
   cin >> a >> b;
   cout << a << ' ' << b << endl;

   c = cin.get();
   cin.get(d);

   cout << c << ' ' << d << endl;

   e = getchar();
   putchar(e);

   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 ниже.

2.4 Стандартные функции работы со троками

2.4.1 <string.h> (<cstring>) – функции работы со строками в стиле C

```
void *memchr(const void *p, int ch, size_t n);
```

Ищет первое вхождение символа в блок памяти.

Функция возвращает указатель на первое вхождение байта, представленного младшим байтом аргумента ch в блоке памяти p длиной n.

```
int memcmp(const void *p1, const void *p2, size_t n);
```

Сравнивает блоки памяти

Функция сравнивает два блока памяти и возвращает значение: меньше нуля, равное нулю или больше нуля – аналогично кодам возврата функции `strcmp`.

```
void *memcpy(void *dest, const void *src, size_t n);
```

Копирует блок памяти

Функция копирует блок памяти длиной n байт из адреса src по адресу dest.

```
void *memmove(void *dest, const void *src, size_t n);
```

Переносит блок памяти

Функция аналогична memcpy, но блоки dest и src могут перекрываться.

```
void *memset(const void *p, int ch, size_t n);
```
Заполняет блок памяти символом

Функция заполняет блок памяти символом, взятым из младшего байта ch.

```
char *strcat(char *s1, char *s2);
```
Складывает строки

Функция добавляет s2 к s1 и возвращает s1. В конец результирующей строки добавляется нуль-символ.

```
char *strchr(char *s, int ch);
```
Ищет символ в строке

Функция возвращает указатель на первое вхождение символа ch в строку s, если его нет, то возвращается NULL.

```
int strcmp(char *s1, char *s2);
```
Сравнивает строки

Функция сравнивает строки и возвращает отрицательное (если s1 меньше s2), нулевое (если s1 равно s2) или положительное (если s1 больше s2) значение.

```
char *strcoll(char *s1, char *s2);
```
Сравнивает строки с учетом установленной локализации

Функция сравнивает строки аналогично `strcmp`, но учитывает установки локализации.

```
char *strcpy(char *s1, char *s2);
```
Копирует одну строку в другую

Функция копирует s2 в s1 и возвращает s1.

```
size_t strcspn(char *s1, char *s2);
```
Ищет один из символов одной строки в другой

Функция возвращает значение индекса любого из символов из s2 в строке s1.

```
char *strerror(size_t n);
```
Возвращает указатель на строку с описанием ошибки

Функция возвращает указатель на строку с описанием ошибки номер n.

```
struct tm strftime(char *s, size_t size, fmt, const struct tm *ctm);
```
Преобразует время в формате fmt в формат tm

Функция возвращает отформатированную строку с датой и временем на основе формата fmt. Значение функции имеет тип time_t, соответствующий типу tm.

```
size_t strlen(char *s);
```
Возвращает длину строки

Функция возвращает длину строки (без учета символа завершения строки).

```
char *strncat(char *s1, char *s2, size_t n);
```
Складывает одну строку с n символами другой

Функция добавляет не более n символов из s2 к s1 и возвращает s1. Первый символ s2 пишется на место завершающего нуль-символа строки s1. Если длина строки s2 меньше n, переписываются все символы s2. К строке s1 добавляется нуль-символ. Если строки перекрываются, поведение не определено.

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

Сравнивает одну строку с n символами другой

Функция сравнивает первую строку и первые n символов второй строки и возвращает отрицательное (если s1меньше s2), нулевое (если s1 равно s2) или положительное (если s1 больше s2) значение.

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

Копирует первые n символов одной строки в другую

Функция копирует не более n символов из s2 в s1 и возвращает s1. Если длина исходной строки превышает или равна n, нуль-символ в конец строки s1 не добавляется. В противном случае строка дополняется нуль-символами до n-го символа. Если строки перекрываются, поведение не определено.

```
char *strpbrk(char *s1, char *s2);
```

Ищет один из символов одной строки в другой

Функция возвращает указатель на символ, являющийся первым вхождением любого из символов из s2 в строку s1, если его нет, возвращается NULL.

```
char *strrchr(char *s,int ch);
```

Ищет символ в строке

Функция возвращает указатель на первое вхождение символа ch в строку s справа, если его нет, возвращает NULL.

```
size_t strspn(char *s1, char *s2);
```

Ищет символ одной строки, отсутствующий в другой

Функция возвращает индекс первого символа в s1, отсутствующего в s2.

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

Ищет подстроку в строке

Функция выполняет поиск первого вхождения подстроки s2 в строку s1. В случае удачного поиска, возвращает указатель на элемент из s1, с которого начинается s2, и NULL в противном случае.

```
double strtod(const char *str, char **end);
```

Преобразует строку в число

Функция преобразует строку символов в числовое значение и возвращает его. При переполнении возвращает +/-HUGE_VAL При невозможности выполнить преобразование или исчезновении порядка возвращает 0. В обоих последних случаях errno устанавливается в ERANGE. end указывает на символ, на котором преобразование завершается.

```
char *strtok(char *s1, char *s2);
```

Выделяет из строки лексемы

Функция возвращает следующую лексему из s1, отделенную любым из символов из набора s2.

```
double strtol(const char *str, char **end, int radix);
```

Преобразует строку в число с учетом системы счисления

Функция преобразует строку символов в числовое значение с учетом указанной системы счисления radix и возвращает полученное число. Функция пропускает возможные начальные пробелы и заканчивает преобразование на первом символе, который не может появиться в образе числа. Параметр end является адресом указателя типа char*; этот указатель будет содержать адрес первого непреобразованного символа. При переполнении возвращает LONG_MAX или LONG_MIN. При невозможности выполнить преобразование возвращает 0. В обоих последних случаях errno устанавливается в ERANGE.

```
double strtoul(const char *str, char **end, int radix);
```

Преобразует строку в число с учетом системы счисления

Функция работает аналогично `strtol`, но работает с беззнаковым длинным целым. При переполнении возвращает ULONG_MAX.

```
size_t strxfrm(char *s1, char *s2, size_t n);
```

Преобразует строки на основе текущей локализации

Функция преобразует строку из s2 и помещение ее в s1 на основе текущей локализации. Преобразуется не более n символов.

2.4.2 <stdio.h> (<cstdio>) – функции ввода-вывода в стиле C

```
int snprintf(wchar_t *buffer, const wchar_t *format[, argument, …]);
```

Выводит строку параметров в определенном формате

Функция выводит в строку buffer значения переменных, перечисленных в списке, обозначенном многоточием, в формате, определенном строкой format. Является аналогом функции `sprintf` для многобайтных символов.

```
int swscanf(const wchar_t *buf, const wchar_t *format, ...);
```

Вводит данные из строки

Функция аналогично функции scanf вводит данные, но не с клавиатуры, а из строки символов, переданной ей первым параметром. Аргумент buf – строка символов, из которой вводятся значения, format – строка формата, в соответствии с которой происходит преобразование данных, а многоточие указывает на наличие необязательных аргументов, соответствующих адресам вводимых значений. Является аналогом функции `sscanf` для многобайтных символов.

```
int sprintf(char *buffer, const char *format[,argument, …]);
```

Выводит строку параметров в определенном формате

Функция выводит в строку buffer значения переменных, перечисленных в списке, обозначенном многоточием, в формате, определенном строкой format.

```
int sscanf(const char *buf, const char *format [,par1, par2, … ]);
```

Вводит данные из строки

Функция аналогично функции scanf вводит данные, но не с клавиатуры, а из строки символов, переданной ей первым параметром. Аргумент buf – строка символов, из которой вводятся значения, format – строка формата, в соответствии с которой происходит преобразование данных, а многоточие указывает на наличие необязательных аргументов, соответствующих адресам вводимых значений.

2.4.3<ctype.h> (<cctype>) – функции классификации и преобразования типов

```
int tolower(int ch);
```

Возвращает символ в нижнем регистре

Функция получает параметр ch и возвращает его в нижнем регистре. В параметре ch используется только младший байт.

```
int toupper(int ch);
```

Возвращает символ в верхнем регистре

Функция получает параметр ch и возвращает его в верхнем регистре. В параметре ch используется только младший байт.

```
int towlower(wint_t ch);
```

Возвращает символ в нижнем регистре

Функция получает символ ch и возвращает его в нижнем регистре. Является аналогом функции `tolower` для многобайтных символов.

```
int towupper(wint_t ch);
```

Возвращает символ в верхнем регистре

Функция получает символ ch и возвращает его в верхнем регистре. Является аналогом функции `toupper` для многобайтных символов.

```
int isalnum(int ch);
```

Проверяет, является ли символ буквой или цифрой

Функция выделяет младший байт параметра ch и возвращает значение true, если символ ch является буквой или цифрой, или false в противном случае.

```
int isalpha(int ch);
```

Проверяет, является ли символ буквой

Функция выделяет младший байт параметра ch и возвращает значение true, если символ ch является буквой, или false в противном случае.

```
int iscntrl(int ch);
```

Проверяет, является ли символ управляющим

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является управляющим символом (типа line feed, del, табуляции и тому подобных, большинство из которых находятся в диапазоне 0x01 – 0x1F (для кодировки ASCII)), или **false** в противном случае.

```
int isdigit(int ch);
```

Проверяет, является ли символ цифрой

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является цифрой, или **false** в противном случае.

```
int isgraph(int ch);
```

Проверяет, является ли символ видимым

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является видимым (то есть он не является символом пробела, табуляции и т. д.) или **false** в противном случае.

```
int islower(int ch);
```

Проверяет, является ли символ буквой нижнего регистра

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является буквой нижнего регистра, или **false** в противном случае.

```
int isprint(int ch);
```

Проверяет, является ли символ печатаемым

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является печатаемым (isgraph + пробел), или **false** в противном случае.

```
int ispunct(int ch);
```

Проверяет, является ли символ символом пунктуации

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является символом пунктуации (то есть печатаемым, но не буквой, не цифрой, не пробелом), или **false** в противном случае.

```
int isspace(int ch);
```

Проверяет, является ли символ разграничительным

Функция выделяет младщий байт параметра ch и возвращает значение **true**, если символ ch является символом пробела или табуляцией, или символом новой строки, или символом новый страницы (символом перевода формата), или **false** в противном случае.

```
int isupper(int ch);
```

Проверяет, является ли символ буквой верхнего регистра

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является буквой верхнего регистра, или **false** в противном случае.

```
int iswalnum(wint_t ch);
```

Проверяет, является ли символ буквой или цифрой

Функция возвращает значение **true**, если символ ch является буквой или цифрой, или **false** в противном случае. Является аналогом функции `isalnum` для многобайтных символов.

```
int iswalpha(wint_t ch);
```

Проверяет, является ли символ буквой

Функция возвращает значение **true**, если символ ch является буквой, или **false** в противном случае. Является аналогом функции `isalpha` для многобайтных символов.

```
int iswcntrl(wint_t ch);
```

Проверяет, является ли символ управляющим

Функция возвращает значение **true**, если символ ch является управляющим символом (типа line feed, del, табуляции и тому подобных, большинство из которыхнаходятся в диапазоне 0x01 — 0x1F (для кодировки ASCII)), или **false** в противном случае. Является аналогом функции `iscntrl` для многобайтных символов.

```
int iswctype(wint_t c, wctype_t desc);
```

Проверяет многобайтный символ

Функция возвращает ненулевое значение, если символ c обладает свойством desc, или нулевое в противном случае.

```
int iswdigit(wint_t ch);
```

Проверяет, является ли символ цифрой

Функция возвращает значение **true**, если символ ch является цифрой, или **false** в противном случае. Является аналогом функции `isdigit` для многобайтных символов.

```
int iswgraph(wint_t ch);
```

Проверяет, является ли символ видимым

Функция возвращает значение **true**, если символ ch является видимым (то есть он не является символом пробела, табуляции и т. д.) или **false** в противном случае. Является аналогом функции `isgraph` для многобайтных символов.

```
int iswlower(wint_t ch);
```

Проверяет, является ли символ буквой нижнего регистра

Функция возвращает значение **true**, если символ ch является буквой нижнего регистра, или **false** в противном случае. Является аналогом функции `islower` для многобайтных символов.

```
int iswprint(wint_t ch);
```

Проверяет, является ли символ печатаемым

Функция возвращает значение **true**, если символ ch является печатаемым (iswgraph + пробел), или **false** в противном случае. Является аналогом функции `isprint` для многобайтных символов.

```
int iswpunct(wint_t ch):
```

Проверяет, является ли символ символом пунктуации

Функция возвращает значение **true**, если символ ch является символом пунктуации (то есть печатаемым, но не буквой, не цифрой, не пробелом), или **false** в противном случае. Является аналогом функции `ispunct` для многобайтных символов.

```
int iswspace(wint_t ch);
```

Проверяет, является ли символ разграничительным

Функция возвращает значение **true**, если символ ch является символом пробела или табуляцией, или символом новой строки, или символом новой страницы (символом перевода формата), или **false** в противном случае. Является аналогом функции `isspace` для многобайтных символов.

```
int iswupper(wint_t ch);
```

Проверяет, является ли символ буквой верхнего регистра

Функция возвращает значение **true**, если символ ch является буквой верхнего регистра, или **false** в противном случае. Является аналогом функции `isupper` для многобайтных символов.

```
int iswxdigit(wint_t ch);
```

Проверяет, является ли символ символом

Функция возвращает значение **true**, если символ ch является символом шестнадцатеричной цифры (цифры, а также буквы от A до F в нижнем или верхнем регистрах), или **false** в противном случае. Является аналогом функции `isxdigit` для многобайтных символов.

```
int isxdigit(int ch);
```

Проверяет, является ли символ символом шестнадцатеричной цифры

Функция выделяет младший байт параметра ch и возвращает значение **true**, если символ ch является символом шестнадцатеричной цифры (цифры, а также буквы от A до F в нижнем или верхнем регистрах), или **false** в противном случае.

3 Контрольные вопросы

1. Какие виды строк существуют в C++?
2. Как объявить C-строку?
3. Как осуществляется ввод-вывод строк?
4. Какие операции над строками вы знаете?
5. Перечислите операции над символами?

4 Задание

1. Написать программу в соответствии с вариантом задания из пункта 5. Для первых двух заданий использовать для работы со строками массив типа char. Для последних двух заданий для работы со строками использовать тип System.String.
2. Отладить и протестировать программы.

5 Варианты заданий

Варианты заданий определяться по последней цифре в списке студентов:
1 вариант 1,11,21,31 задание.
2 вариант 2,12,22,32 задание.
3 вариант 3,13,23,33 задание.
4 вариант 4,14,24,34 задание.
5 вариант 5,15,25,35 задание.
6 вариант 6,16,26,36 задание.
7 вариант 7,17,27,37 задание.
8 вариант 8,18,28,38 задание.
9 вариант 9,19,29,39 задание.
0 вариант 10,20,30,40 задание.

1) Написать программу которая преобразует строку таким образом, что цифры, которые находятся в слове, переносятся в конец строки без изменения порядка следования остальных символов.
2) Написать программу, которая, если строка начинается и оканчивается одним и тем же знаком, во всей строке заменяет этот знак четвертым символом строки.
3) Дана строка состоящая из слов, разделенных пробелами (одним ил несколькими). Вывести строку содержащую эти же слова (разделенные одним пробелом), но расположенные в обратном порядке.
4) Дана строка символов. Для сохранения ее в сжатом виде найти максимальную последовательность символов произвольной длины, которая повторяется, и заменить ее своим кодом.
5) В строку S добавить необходимое количество пробелов так, чтобы ее длина стала равна n. Причем: перед первым словом пробелы не добавлять, после последнего слова все пробелы удалить, добавленные пробелы равномерно распределить между словами. Если длина S превосходит n, удалить S из все слова, которые не укладываются в первые n символов, а оставшуюся часть преобразовать по вышеуказанным правилам.
6) Палиндромом называют последовательность символов, которая читается как слева направо, так и справа налево. Найти во введённой строке подстроку-палиндром максимальной длины.
7) Вывести сообщение "МОЖНО", если из букв введённой строки X можно составить введенную строку Y, при условии, что каждую букву строки X можно использовать один раз; и сообщение "НЕЛЬЗЯ" в противном случае.
8) По введённому числовому значению N ($0 < N < 4000$) вывести его запись в римской системе счисления. Римская система счисления использует 7 цифр (I=1 V=5 X=10 L=50 C=100 D=500 M=1000).
9) Ввести n слов с консоли. Найти слово, в котором число различных символов минимально. Если таких слов несколько, найти первое из них.
10) Ввести n слов с консоли. Найти количество слов, содержащих только символы латинского алфавита, а среди них – количество слов с равным числом гласных и согласных букв.
11) Ввести n слов с консоли. Найти слово, символы в котором идут в строгом порядке возрастания их кодов. Если таких слов несколько, найти первое из них.

12) Ввести n слов с консоли. Найти слово, состоящее только из различных символов. Если таких слов несколько, найти первое из них.
13) Заменить в данной строке все вхождения подстроки s на порядковый номер вхождения. Подстрока s вводится с клавиатуры.
14) Ьтасипан уммаргорп адовереп йоннадаз икортс оп умещюуделс упицнирп.
15) В строке переставить местами рядом стоящие слова.
16) Из строки вырезать слова, стоящие на четном месте.
17) В строке удалить все пробелы, а затем после каждой пятой буквы вставить знак вопроса.
18) Переставить местами слова в строке (первая с последней и т.д.).
19) Из строки удалить все встречающиеся символы.
20) Все слова в строке расположить в алфавитном порядке.
21) Подсчитать сколько раз в данной строке встречается некоторая последовательность букв, введенная с клавиатуры.
22) В строке после каждого слова вставить запятую.
23) Каждое слово в строке распечатать с новой строчки экрана.
24) В строке удалить последнюю букву у слов.
25) В строке все запятые заменить точкой, и перед первым словом вставить слово STRING.
26) Подсчитать количество слов и букв в этих словах в строке.
27) В строке удалить все знаки препинания.
28) С клавиатуры считывается строка состоящая из цифр от 0 до 9. Разбить ее на две части, полученные строки преобразовать к целочисленному типу.
29) В строке каждый символ заменить на соответствующий ему код.
30) В строке удалить каждый заданный символ, а остальные продублировать.
31) В строке посчитать наибольшее количество идущих подряд пробелов.
32) В строке удалить лишние пробелы.
33) В строке подсчитать количество слов начинающихся с заданной буквы.
34) В строке удалить все символы не являющиеся буквами или цифрами.
35) В строке найти самое длинное симметричное слово.
36) В строке оставить только те символы которые встречаются один раз.
37) В строке указать слово, в котором количество гласных букв минимально.
38) В строке указать слово, в котором количество согласных букв максимально.
39) В строке удалить все заданные группы букв.
40) В строке подсчитать количество слов