Для удаления warning

#pragma warning(disable:4996)

или

#define \_CRT\_SECURE\_NO\_WARNINGS

# **Упр 1 Определить результат работы**

# **int \*x; x= int new [3]; \*x=1; x++; \*x=2; x++; \*x=3; cout<<x[-2]<<' '<<x[-1]<<' '<<x[0]<<endl; cout<<x[0]<<' '<<x[1]<<' '<<x[2]<<endl; cout<<\*(x-2)<<' '<<\*(x-1)<<' '<<\*x<<endl; cout<<\*x--<<' '<<\*x--<<' '<<\*x--<<endl; cout<<x[0]<<' '<<x[1]<<' '<<x[2]<<endl;**

# **cout<<x[-2]<<' '<<x[-1]<<' '<<x[0]<<endl; 1 2 3 cout<<x[0]<<' '<<x[1]<<' '<<x[2]<<endl; 3 1832195752 1832196136 cout<<\*(x-2)<<' '<<\*(x-1)<<' '<<\*x<<endl; 1 2 3 cout<<\*x--<<' '<<\*x--<<' '<<\*x--<<endl; 3 3 3 cout<<x[0]<<' '<<x[1]<<' '<<x[2]<<endl; 335590640 1 2**

# **11. Обработка строк** **11.1. Строковые константы** В Си++ различают символьные и строковые константы и переменные. Символьная константа – это символ в одинарных кавычках. Она имеет тип char. Строковая константа – это последовательность символов кода ASCII, заключённая: "...". Она имеет тип char[].

# Примеры: 'y' 's' '#' "T" "This is \" character string" "This is character string" "Это строковая константа" "A" "1234567890" "0" "$" В конце каждой строки компилятор помещает нулевой символ '\0'.

# **11.2 Определение и инициализация строк.**  Строкой называется массив символов, который заканчивается символом '\0' (нулевой байт). Поэтому строка объявляется как обычный символьный массив: **char \*str; char str[10];** При инициализации строки размерность массива лучше не указывать. **char \*str="This is a string"; char str[]="This is a string";**

# Этот вид инициализации является краткой формой стандартной инициализации массива char str[]={'T','h', 'i', 's', '\0'}; Без символа 0 мы имеем массив символов, а не строку. Имя **str** является указателем на первый элемент массива: cout<<str; //This cout<<&str[0]; //This cout<<\*str; //T cout<<\*(str+1); //h

# **11. 3. Массив и указательchar mhead[ ] = "Программирование на Cи++"; char \*uhead = "Программирование на Pascal";** Основное отличие состоит в том, что указатель **mheard** является **константой,** в то время как указатель **uhead - переменной**. В том и в другом случае можно использовать операцию сложения с указателем:

# **for( int i=0;i<16;i++) cout<<\*(mhead+i)) <<endl;   for(i=0;i<16;i++) cout<<\*(uhead+i))<<endl;  В результате получаем Программирование Программирование**

# Операцию увеличения можно использовать только с указателем: **while ((\*uhead) != '\0') cout<<\*(head++); *Программирование******на языке Cи++*** **💣****while ((\*mhead) != '\0') cout<<\*(mhead++);** Можно так: **uhead=mhead;** Так нельзя **mhead = uhead**; В данном случае uhead = mheard, не уничтожит строку про язык Cи, а только изменит адрес, записанный в uhead.

# **11.4.Указатели и строки**Большинство операций языка Си, имеющих дело со строками, работают с указателями. main( ) { char \*mesg = "Message"; char \*copy; copy = mesg; } **При присваивании указателей строка никогда не копируется. Оператор copy = mesg; создает второй указатель, ссылающийся на ту же самую строку.**

# **11.5. Ввод-вывод строк char \*gets (char \* s); puts (char\*s); gets -** прочитать строку из стандартного файла ввода **stdin. puts -** записать строку в стандартный файл вывода **stdout.** В конце строк записывается символ новой строки. **char s[10]; // char \* s; gets(s); cin>>s; puts(s); cout<<s;**

# **11.6. Обработка строкСтрока символов хранится в памяти как массив и заканчивается нулевым символом, доступ к ней осуществляется с помощью указателя типа char. Указатель указывает на соответствующий строке массив символов. Для указателя на строковую константу, например const char \*st=”English\n”; компилятор помещает все символы строки в массив и затем присваивает переменной st адрес первого элемента массива. Для перебора символов строки используется адресная арифметика, например так: While (\*st++) {…}**

# **Эта функция определяет длину строки.**  **int string\_length (const char \*st)** { int length\_st=0; if (st) while (\*st++) ++length\_st; return length\_st; }

# **int str\_length (const char \*st)** { int length\_st=0; if (st) for ( int i=0; st[i]!='\0'; i++) length\_st++; return length\_st; }

# **Стандартные функции для обработки строк. #include <string.h>**

# **11.6.1.strcat - сцепить две строки**. Для соединения строк в одну используются: **char\* strcat(char \*str1, const char \*str2);** Присоединяет str1 к str2, завершающий нулевой байт в str1 стирается, возвращает указатель на str1. **Пример 1**:/\* сцепить две строки \*/ **#include <string.h>** char dest[25]= "Программирование " ; char \*blank = " ", \*c = "C++"; char \*res; strcat(dest, blank); strcat(dest, c);

# **11.6.2.strncat - сцепить две строки, не более n- символов из второй. char\* strncat(char \*str1,const char \*str2, size\_t n);** Присоединяет n символов из str2 к str1, завершающий нулевой байт str1 стирается. Если len(str2)<n, то присоединяется только символы входящие в str2. После присоединения к str1 добавляется "\0" **Пример 2:** { **char \*dest= new [25]; char \*source = "Visual С++"; strcpy(dest, "program"); strncat(dest, source, 6);}**

# **11.6.3. strcmp - сравнить две строки в лексикографическом порядке. int strcmp(char \*str1, const char \*str2)** Если \*str1 в лексикографическом порядке идет после \*str2, то strcmp возвращает число, большее нуля. Если две строки совпадают, то strncmp возвращает ноль. Если \*str1 в лексикографическом порядке идет пеpед \*str2, то strcmp возвращает число, меньшее нуля.

# **Пример 3:** char \*buf1 = "aaa", \*buf2 = "bbb"; int ptr; ptr = strcmp(buf2, buf1); **if (ptr == 0) cout<<" 2 is equal than 1\n"; else if (ptr > 0) cout<<" 2 is greater than 1\n"; else cout<<“ 2 is less than 1\n"; }**

# **11.6.4. strncmp - сравнить первые n символов двух строк. int strncmp**(**char \*str1, const char \*str2, size\_t n);** Пример 4: { char \*buf1 = "aaabbb", \*buf2 = "bbbccc"; int ptr; ptr = strncmp(buf2, buf1,4); if (ptr == 0) cout<<" 2 is equal than 1\n"; else if (ptr > 0) cout<<" 2 is greater than 1\n"; else cout<<"2 is less than 1\n"; }

# **11.6.5. strcpy - копировать строку str2 в строку str1.** **char \*strcpy (char \*str1, const char \*str2);** Копирует str2 в str1 (str2 копируется включая "\0"). Возвращает указатель на str1. Если строки перекрываются, то результат не предсказуем.  Пример 5: void main() { char string[10]; char \*str1 = "abcdefghi";   strcpy(string, str1); }

# **11.6.6. strncpy - копировать не более n символов строки str2. char\* strncpy(char \*str1, const char \*str2, size\_t n);** Копирует n символов из str2 в str1. Если в str2 символов меньше, чем n, то последний нулевой байт дополняется до n символов. Возвращается указатель на str1. Пример 6: { **char string[10]; char \*str1 = "abcdefghi"; strncpy(string, str1, 3); string[3] = '\0'; }**

# **11.6.7. strlen - определить длину строки (число символов без завершающего нулевого символа). size\_t strlen(const char \*str);** возвращает длину строки не учитывая нулевой байт. Пример 7: **{ int len; char \*str = "Программирование!"; len=strlen(str); //17 }**

# **11.6.8. strchr - найти первое вхождение символа с. char\* strchr(const char \*str, int c);** Ищет первое вхождение символа с в str, в случае успеха возвращает указатель на первый найденный символ, если неудача =NULL; Пример 8: **{ char str[20]; char \*ptr, c = 'r'; strcpy(str, "This is a string"); ptr = strchr(str, c); if (ptr) cout<<«Адрес символа: “<< ptr; else cout<<«Символ не найден";}**

# 1245004 адрес начала 1245016 адрес символа **r** **int n=ptr-str; cout<<“ Позиция символа “<<c <<“=“ <<n; Позиция символа r =12** **cout<<\*ptr<<endl; //вывод символа**

# **11.6.9. strrchr - найти в строке последнее вхождение символа ( обратный поиск знака в строке ) char\* strrchr(const char \*str, int c);** Ищет последнее вхождение с в str (преобразованного в char) char \*s; int c; **Пример 9:** void main() { char string[20]; char \*ptr, c = 's'; strcpy(string, "This is a string"); ptr = strrchr(string, c); }//адрес S string

# **11.6.10. strpbrk -** найти в строке str1 первое появление любого из множества символов, входящих в строку str2 ( исключая завершающий знак NULL), возвращает yказатель на найденный в str1 символ или NULL-yказатель, если знаков из str2 в str1 нет. **char\* strpbrk(const char \*str1, const char \*str2); char str[80]="123 abc"; char \*st1, \*st2; st1=strpbrk(str,"bca"); st2=strpbrk(str,“1bca"); //st1= abc //st2 =123 abc**

# **Пример 11.1 #include <string.h> void main(void) { char \*string1 = "abcdemnopqrstu"; char \*string2 = "onm"; int \*ptr; ptr = strpbrk(string1, string2); if (ptr) // адрес m cout<<ptr; //mnopqrstu else cout<< “no”;}**

# **11.6.11. strspn** Функция strspn() возвращает длину начальной подстроки строки, адресуемой параметром *str1*, которая состоит только из символов, содержащихся в строке, адресуемой параметром *str2*. Другими словами, **функция strspn() возвращает индекс первого символа в строке *str1*, который не совпадает ни с одним из символов в строке *str2***

# **Пример 11.2 Ищем string1 в st1** {char \*string1 = "1234567890"; char \*st1 = "123DC8", \*st2="123D45C8"; char \*st3 = "123DC584176"; int len1,len2, len3; int len1 = strspn(string1, st1); //1234567890 int len2 = strspn(string1, st2); //1234567890 int len3 = strspn(string1, st3); } //1234567890 // len1=3 len2=5 len3=8

# **11.6.12. strcspn size\_t strcspn(const char \*str1, const char \*str2 );** Функция strcspn() возвращает количество символов в начале строки с указателем str1, среди которых нет ни одного символа, входящего в строку с указателем str2. **Для строки str1 функция strcspn() возвращает значение индекса первого символа из числа символов, входящих в строку str2.**

# **Пример 11.3 В строке string1 ищем номер первого символа, который есть строку string2 . void main(void) { char \*string1 = "1234567890"; char \*string2 = "747DC8"; char \*string3 = "7471DC8"; int len1,len2; len1 = strcspn(string1, string2); len2 = strcspn(string1, string3); // len1= 3 len2= 0**

# **11.6.13. strtok - выделить из строки s1 лексемы, разделенные любым из множества символов, входящих в строку s2. char\* strtok(const char \*str1, const char \*str2);** возвращает указатель на следующую лексему. Последовательность символов) строки str1, разделителями лексем в котором является символы строки str2. Возвращает NULL если больше нет лексем. При первом вызове в качестве str1 передается адрес строки. В последующих вызовах для этой строки в качестве str1 передается NULL. После нахождения лексемы функция strtok вместо разделителя записывает "\0".

# **Пример 11.4 void main() { char input[16] = "abc,d"; char \*p; p = strtok(input, ","); if (p) cout<< p; p = strtok(NULL, ","); if (p) cout<< p; }**

# **Пример 11.5** void RStrtok(void) **// разделение строки на лексемы (слова)** { char string[] = "A string\tof ,,tokens\nand some more tokens"; char seps[] = " ,\t\n"; char \*token;  cout<< "Tokens:\n" ; // поиск первого слова: token = strtok( string, seps );

# while( token != NULL ) { cout<< token;   // выбор следующего слова: token = strtok( NULL, seps ); } }

# **11.6.14. strstr - Находит первое вхождение str2 в str1. char\* strstr(const char \*str1, const char \*str2);** Находит первое вхождение str2 в str1 (без "\0"). В случае успеха возвращает указатель на найденную подстроку, в случае неудачи – 0. Если str1 указывает на строку нулевой длины, то возвращает указатель на str1. **char str[10]=”123 abc 456”; char \*st; st=strstr(str,”abc”); cout<<st; //печать: abc 456**

# **11.6.15. Функции для работы с памятью. void \*memchr(const void \*str, int c, size\_t n); Ищет первое вхождение с в n байтах str. int memcmp(conct void \*str1, const void \*str2, size\_t n); Сравнивает первые n байт str1 и str2. void \*memcpy(void \*str1, const void \*str2,size\_t n); Копирует n байт из str2 в str1 void \*memmove(void \*str1, const void \*str2,size\_t n); Копирует n байт из str2 в str1, перекрыв. строк void \*memset(void \*str1, int c,size\_t n); Копирует c в n байт в str1**.

# **11.6.16. Функции проверки** **на принадлежность #include <ctype.h> int isalnum(int ch);**

# **isalnum —** Проверка на принадлежность символа к алфавитно-цифровым **isalpha —** Проверка на принадлежность символа к буквам **isblank —** Проверка пустого символа **iscntrl —** Проверка на принадлежность символа к управляющим **isdigit —** Проверка на принадлежность символа к цифровым **isxdigit** — Проверка на принадлежность символа к шестнадцатеричным

# **isgraph —** Проверка на принадлежность символа к печатным но не к пробелу **islower —** Проверка на принадлежность символа к строчным **isprint —** Проверка на принадлежность символа к печатным **ispunct —** Проверка на принадлежность символа к знакам пунктуации **isspace —** Проверка на принадлежность символа к пробельным **isupper —** Проверка на принадлежность символа к прописным

# **Тема 12. Функции преобразования строк 12.1. Функции преобразования строки в числовые данные.  Определены в <stdlib.h> int atoi (const char \*str); long int atol (const char \*str); double atof(const char \*str);**

# На вход они принимают указатель на строку, завершенную нулем, а возвращают - число, которое этой строкой описывается. **atoi** и **atol** воспринимают следующий формат числа: [*пробелы*][*знак*]*цифры* **atof**, соответственно: [*пробелы*][*знак*][*цифры*][**.***цифры*][{**d** | **D** | **e** | **E** }[*знак*]*цифры*]

# **char str[]="12345"; int x,y; double z; x = atoi ("12345"); y = atol (str); z = atof ("12.34"); Основной недостаток этих функций заключается в том, что они никак не сигнализируют об ошибке, если таковая произошла в процессе разбора переданной строки - несоответствие его формату или по иным причинам.**

# **Эту проблему решает следующий набор библиотечных функций, также включенных в стандартную библиотеку: long strtol (const char\* str, char\*\* end\_ptr, int radix) unsigned long strtoul (const char\* str, char\*\* end\_ptr, int radix) double strtod (const char\* str, char\*\* end\_ptr)**

# { int n; char \*p; **n=strtol("12a",&p,0);** printf("n=%ld, stop %c\n",n,\*p); **n=strtol("0x12",&p,0);** printf("n=%ld, stop %c\n",n,\*p); **n=strtol("01117",&p,2);** printf("n=%ld, stop %c\n",n,\*p); return 1;} //n=12 stop=a //n=18 //n=7 stop=7

# **unsigned long int strtoul(...); double strtod(const char \*str, char \*\* endptr);** Замечание 1: все функции перечисленные в этом параграфе, прекращают свою работу при встрече символа, который не подходит под параметры этого числа. Замечание 2: в случае если символ представлениия превосходит допустимый диапазон, то функции atof, strol, strtoul, strtod устанавливают переменную errno в значение ERANGE(в <math.h>), при этом функции atof и strtod возвращают значение HUGE\_VAL, функция strtol возвращает long\_max или long\_min, а функция strtoul возвращает long\_max.

# **12.2 Функции преобразования числовых данных встроку** Нет специализированной функции (описанной в стандарте языка) для обратного преобразования (числа в строку). Но такие функции есть в RTL компиляторах.

# **12.2.1 Преобразование целого числа в строку** Для преобразования - следующими функциями: **char\* \_itoa(int value, char\* string, int radix); char\* \_ltoa(long value, char\* string, int radix); char\* \_ultoa(unsigned long value, char\* string, int radix); эти функции входят в библиотеку компиляторов от Microsoft.**

# ss= \_itoa(12345,ss, 10); //12345 ss= \_itoa(12345,ss, 8); //30071 ss= \_itoa(12345,ss, 2); //11000000111001

# **12.2.2 Преобразование числа с плавающей точкой в строку** Для перевода чисел с плавающей точкой - функции: **char\* \_ecvt(double value, int count, int\* dec, int\* sign); char\* \_fcvt (double value, int dcount,int\* dec, int\* sign); char\* \_gcvt (double value, int digits, char\* buffer); - для Microsoft-компиляторов**

# **double val= -12446.6; int count=10; int dec; int sign; char\* ss; ss = (char\*)malloc(10); ss=\_ecvt(val, count, &dec, &sign);**

# Назначение функций следующее: Функция **ecvt** конвертирует число (value) в заданное (count) количество символов, не выполняя при этом никакого форматирования. Через параметр dec возвращается позиция, с которой начинается дробная часть (позиция десятичной точки), начиная с первого символа строки, а через параметр sign - признак того, что число имеет знак. Замечания:

# 1.      Если строковое представление числа уже, чем необходимое количество символов, то число справа дополняется нулями. 2.      Если строковое представление числа шире, чем необходимое количество символов, то возвращаемая через dec позиция находится правее завершающего строку нуля или левее первого символа строки. Выбор знака dec зависит от знака десятичной экспоненты числа. Например, для value= 3.5678e20 и count=6 функция вернет строку "356780", а dec будет равно 21. А для значения 3.5678e-20 и 6, будет возвращено, соответственно, "356780" и -19. 3.      Преобразование производится во внутренний статический буфер, разделяемый с функцией **fcvt**, по этому необходимо быть осторожным при использовании функции в многопоточной среде.

# Функция **fcvt** отличается от **ecvt** только тем, что параметр count задает не общее число символов, а число символов после десятичной точки (точность представление числа). При этом строковое представление (по необходимости) дополняется справа нулями до получения нужной ширины. Параметры dec и sign ведут себя аналогичным (как и для **ecvt**) образом. Для приведенных выше примеров возвращенные значения будут "356780000000000000000000000" и 21, а также "" и -19 соответственно. В последнем случае возвращенная пустая строка означает, что строковое представление содержит только нули.

# Функция **gcvt** преобразует заданное число (value) в строковое представление и помещает результат в переданный буфер (buffer). При этом в буфер будет помещено не более digits цифр. Если число может быть представ-лено таким количеством цифр, то оно будет преобразовано в десятичный формат, иначе будет выполнено преобразование в экспоненци-альный формат, и заданное количество цифр будет содержать мантисса числа. Строковое представление дополняется справа нулями для получения необходимого количества разрядов.

# **/\*1. Функция для сортировки массива строк.** Строки вводятся с консоли.\*/ #include <iostream.h> #include <string.h> #include <stdio.h> #include <stdlib.h> int N; char \*\*ma; int comp(void\* x,void\* y) { return strcmp(\*(char \*\*)x,\*(char\*\*)y); }

# void main() { cout<<"Input N"<<endl; cin>>N; ma=new char\*[N]); for (int i=0;i<N;i++) ma[i]=new char[N]; cout<<"Input strings\n"; for (i=0;i<N;i++) gets(a[i]); qsort(a,N,sizeof(char\*),(int (\*)(const void\*, const void\*))comp); for (i=0;i<N;i++) puts(ma[i]); }

# void delstr(const char\*A,const char\*B) { char\* s; **//Поиск строки В в строке А** int n=strlen(B); //Длина подстроки while((s=(char\*)strstr(A,B))!=NULL) //Пока в строке есть заданные подстроки: { while(\*(s+n)!=NULL) { //Сдвигаем всю строку влево стирая найденную подстроку **\*s=\*(s+n); s++; }** \*s=NULL; //Ограничиваем ее }}

# bool isLetter(char c) { if (('a'<=c && c<='z') || ('A'<=c && c<='Z')) return true; else return false;