

Символьные данные и строки

Для символьных данных в C++ введен тип **char**.

Описание символьных переменных:

char список_имен_переменных;

Пример 63

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

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.

Количество элементов в таком массиве на 1 больше, чем изображение соответствующей строки, так как в конец строки добавлен нулевой символ '\0' (рис. 7.1).

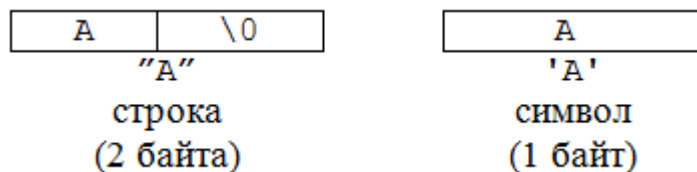


Рис. 7.1. Представление строки и символа

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 64 Определение размера строк

```
#include <iostream>
using namespace std;
int main()
{
    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";
char *s4="string4"; //указатель на строку, ее нельзя изменить
k=sizeof(s4);
cout<<s4<<"\t"<<k<<"\n";
system("pause");
return 0;
}

```

Результат выполнения программы:

string1 10 – выделено 10 байтов, в том числе под '\0'
string2 8 – выделено 8 байтов (7 + 1 байт под '\0')
string3 8 – выделено 8 байтов (7 + 1 байт под '\0')
string4 4 – размер указателя

Если количество символов, присваиваемых строке, меньше размера массива, большинство компиляторов C++ будут присваивать символы NULL, остающимся элементам массива. Как и в случае с массивами других типов, если не указан размер инициализируемого при объявлении массива, компилятор C++ распределит достаточно памяти для размещения указанных букв и символа NULL.

ВВОД-ВЫВОД СИМВОЛЬНЫХ ДАННЫХ И СТРОК

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*.

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 65. Программа считывает из входного потока один символ, а затем выводит его на экран

```

int main()
{
    char ch;
    cout<<"Input text";
    ch=getchar();
    putchar(ch);
    return 0;
}

```

Пример 66. Введите предложение, в конце которого стоит точка, и подсчитайте общее количество символов, отличных от пробела (не считая точки).

```

int main()
{
    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 ранг, чем операция сравнения*/

```
if (z!=' ')  
    k++;  
printf("\nКоличество символов=%d", k);  
return 0;  
}
```

Результат выполнения программы:

Напишите предложение с точкой в конце:

1 2 3 4 5 6 7 8 9 0. МАМА МЫЛА РАМУ

Количество символов=10

2) Ввод-вывод стандартного текстового (символьного) потока

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

puts(s) – функция 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.

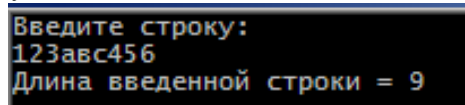
Пример 67.

```
int main()  
{  
    char s[20];  
    cout<<"Input text";  
    gets(s);  
    puts(s);  
    return 0;  
}
```

Результат выполнения программы: при вводе строки "123 456 789" чтение данных осуществляется побайтно до символа '\n', то есть в *s* занесется строка "123 456 789\0" (управляющая последовательность '\0' на экран не выводится, а является признаком конца строки). При выводе строки функция *puts* возвращает в конце строки дополнительно один символ '\n', следовательно, будет выведена строка "123 456 789\n" (управляющая последовательность '\n' на экран не выводится, а осуществляет перевод курсора на новую строку).

Пример 68. Вычислите длину введенной строки.

```
int main()  
{  
    char st[100];  
    int i=0;  
    puts("Введите строку:");  
    gets_s(st);  
    while(st[i++]);  
    printf("Длина введенной строки = %i\n",i-1);  
    system("pause");  
    return 0;  
}
```



```
Введите строку:  
123авс456  
Длина введенной строки = 9
```

3) Стандартные потоки ввода-вывода символьных данных и строк

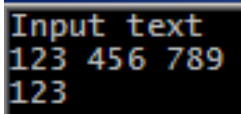
cin – оператор, который определяет стандартные потоки ввода данных.

cout – оператор, который определяет стандартные потоки вывода данных.

<< – операция записи данных в поток;
>> – операция чтения данных из потока.

Пример 69.

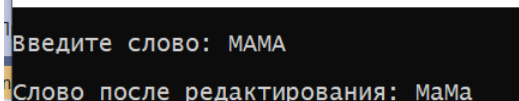
```
int main()
{
    char s[20];
    cout<<"Input text";
    cin>>s;        //ввод строки из стандартного потока
    cout<<s;        //вывод строки в стандартный поток
    return 0;
}
```



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

Пример 70. Введите слово и замените в нем все вхождения заглавной латинской 'A' на малую латинскую 'a'. Выведите слово после редактирования.

```
int main()
{
    SetConsoleOutputCP(1251);
    char st[80];
    int i;
    cout << "\nВведите слово: ";
    cin >> st;
    for(i=0;st[i]!='\0';i++)
        if (st[i]=='A') st[i]='a';
    cout << "\nСлово после редактирования: "<< st;
    return 0;
}
```



4) Форматированный ввод-вывод символьных данных и строк

printf() – функция, осуществляющая форматированный вывод данных.

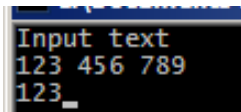
scanf() – функция, осуществляющая форматированный ввод данных.

%c – спецификатор формата ввода-вывода одиночного символа.

%s – спецификатор формата ввода-вывода строки символов.

Пример 70.

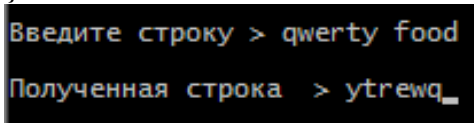
```
int main()
{
    char s[20];
    cout<<"Input text";
    scanf("%s",s);
    //для строк не используется обращение по адресу &
    printf("%s",s);
    return 0;
}
```



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", следовательно, выведется: "123". Так как *s* — имя символьного массива, то есть адрес его начального элемента, операция **&** в функции *scanf* для строк не используется.

Пример 71. Записать введенную строку символов в обратном порядке.

```
int main()
{
    char st[80];
    char temp;
    int i, len=0;
    printf("\nВведите строку > ");
    scanf("%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--)
    { //обмен символов
        temp=st[i];
        st[i]=st[len];
        st[len]=temp;
    }
    printf("\nПолученная строка > %s", st);
    system("pause");
    return 0;
}
```



5) Прочитать несколько строк текста можно так:

```
char str[100];
cin.getline(str, 100, '*');
```

Второй параметр этой функции (100) задаёт максимальную длину строки, третий ('*') — по какому символу прекратить ввод.

```
int main()
{
    char s[200];
    cout<<"Input text\n";
    cin.getline(s, 100, '*');
    cout<<"Result\n";
    printf("%s", s);
    getch();
    return 0;
}
```

```
Input text
char str[100];
cin.getline(str, 100, '*');
Result
char str[100];
cin.getline(str, 100, '

```

Внутренние коды символов

На базовом уровне компьютеры хранят всю информацию в виде цифр. Для представления символьных данных используется схема перевода, которая содержит каждый символ с его репрезентативным номером.

Самая простая схема в повседневном использовании называется ASCII.

ASCII — это таблица кодировки символов, в которой каждой букве, числу или знаку соответствует определенное число. В стандартной таблице ASCII 128 символов, пронумерованных от 0 до 127. В них входят латинские буквы, цифры, знаки препинания и управляющие символы.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(72	48	110	H	104	68	150	h
9	9	11		41	29	51)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

В языке C++ принято соглашение, что всюду, 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. К примеру, внутренние коды десятичных цифр в таблицах кодов ASCII упорядочены по числовому значению, поэтому несложно перебрать символы десятичных цифр в нужном порядке.

Что такое ASCII

Таблицу разработали в Америке в 60-х, и ее название расшифровывается как American Standard Code for Information Interchange — Американская стандартная кодировка для обмена информацией. Аббревиатура читается как «аски».

Существуют национальные расширения ASCII, которые кодируют буквы и символы, принятые в других алфавитах. «Стандартная» таблица называется US-ASCII, или международной версией. В большинстве национальных расширений заменена только часть символов, например знак доллара на знак фунта. Но для языков, где используются нелатинские алфавиты, заменяется большинство символов. Русский относится к таким языкам.

Для чего нужна таблица ASCII

Цифровое устройство по умолчанию не понимает символы — только числа. Поэтому буквы, цифры и знаки приходится кодировать, чтобы задавать компьютеру соответствие между определенным начертанием и числовым значением. Сейчас вариантов кодирования несколько, и ASCII — одна из наиболее ранних кодировок. Она задала стандарты для последующих решений.

Когда появилась эта кодировка, компьютеров в современном представлении еще не существовало. Ее разработали для телетайпов — устройств обмена информацией, похожих на телеграфы с печатной машинкой. Сейчас ими практически не пользуются, но некоторые стандарты остались с тех времен. В том числе набор ASCII, который теперь применяется для кодирования информации в компьютерах.

Сейчас с помощью ASCII кодируются данные в компьютерных устройствах, на ней основано несколько других кодировок, кроме того, ее используют в творчестве — создают с помощью символов картинки. Это называется ASCII art.

Применение на практике

При разработке сайта или приложения разработчику может понадобиться пользоваться ASCII, чтобы закодировать символы, не входящие в национальную кодировку.

Можно сохранить документ или иной файл в формате ASCII — тогда все символы в нем будут закодированы этим набором. Такое может понадобиться, если человеку нужно передать информацию, которая будет читаться везде, — но некоторые функции форматирования в таком режиме будут недоступны.

Можно ввести код ASCII с клавиатуры напрямую: при зажатом Alt набрать числовое значение, которое соответствует тому или иному символу из таблицы. Так можно печатать и символы, которые есть в расширенных версиях набора: смайлики, иероглифы, буквы алфавитов других стран и так далее. Код для таких символов может быть намного длиннее, чем для стандартных 128 букв и цифр.

Как устроена ASCII внутри

С помощью ASCII вводят, выводят и передают информацию, поэтому она должна описывать самые часто используемые символы и управляющие элементы (перенос, шаг назад и так далее). Таблица восьмибитная, а числа, которые соответствуют символам, переводятся в двоичный код, чтобы компьютер мог их распознавать. Десятичное же написание удобнее для людей. Еще используют шестнадцатеричное — с его помощью легче представить набор в виде таблицы.

Заглавные и строчные буквы в ASCII — это разные элементы. Причем в таблице строчные буквы расположены под заглавными, в том же столбце, но в разных строках. Так набор оказывается нагляднее, а информацию легче проверять и работать с ней, например редактировать регистр с помощью автоматических команд.

Как расположены символы в ASCII

- Первые две строки таблицы — управляющие символы: Backspace, перевод строки, начало и конец абзаца и прочие.

- В третьей строке расположены знаки препинания и специальные символы, такие как процент % или астериск *.
- Четвертая строка — числа и математические символы, а также двоеточие, точка с запятой и вопросительный знак.
- Пятая и шестая строчка — заглавные буквы, а также некоторые другие особые символы.
- Седьмая и восьмая строки описывают строчные буквы и еще несколько символов.

Отличия от Unicode

Когда мы говорим о кодировании, сразу вспоминается система международной кодировки символов Unicode. Важно не путать ее с ASCII — эти понятия не идентичны.

ASCII появилась раньше и включает в себя меньше символов. В стандартной таблице их всего 128, если не считать расширений для других языков. А в «Юникоде», который реализуют кодировки UTF-8 и UTF-32, сейчас 2^{21} символов — это больше чем два миллиона. В набор входят практически все существующие сегодня символы, он очень широкий.

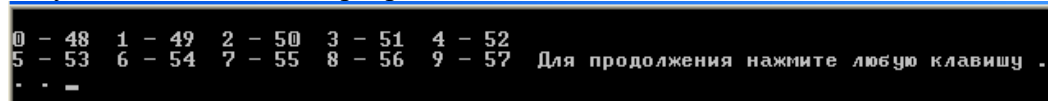
Unicode можно рассматривать как «продолжение», расширение ASCII. Первые 128 символов в «Юникоде» кодируются так же, как в ASCII, и это те же самые символы.

Unicode — это современный стандарт, который пытается предоставить числовой код для всех возможных символов, на всех возможных языках, на каждой возможной платформе.

Пример 72. Печать десятичных цифр и их кодов

```
int main()
{
    char z;
    for (z='0'; z<='9'; z++) {
        if (z=='0' || z=='5')
            printf ("\n");
        printf ("%c - %i ", z, z);
    }
    system("pause");
    return 0;
}
```

Результат выполнения программы:



```
0 - 48  1 - 49  2 - 50  3 - 51  4 - 52
5 - 53  6 - 54  7 - 55  8 - 56  9 - 57  Для продолжения нажмите любую клавишу .
- - -
```

Внутренние коды больших и малых символов латинского алфавита представлены соответственно последовательными величинами. Однако при использовании символов кириллицы необходимо учитывать, что между малыми символами 'п' и 'р' размещены символы псевдографики, прерывающие прохождение кодов. Поэтому использование стандартных функций изменения регистра для символов кириллицы может привести к некорректному результату.

Пример 73. Вывести на экран вторую часть таблицы кодировки символов (символы с кодами от 128 до 255). Таблица должна состоять из восьми колонок и шестнадцати строк.


```

#include <iostream>
using namespace std;
#define SM 128
int main()
{
    unsigned char ch; /*если ch объявить как char, то буквам русского алфавита
                        будут соответствовать отрицательные коды*/

    int i, j;
    printf("Таблица ASCII кодировки символов\n");
    for (i=0; i<16; i++) // шестнадцать строк
    {
        ch=i+SM;
        for (j=1; j<=8; j++) // восемь колонок
        {
            printf ("%4c -%4i",ch,ch);
            ch+=16;
        }
        printf ("\n");
    }
    system("pause");
    return 0;
}

```

Результат выполнения программы:

А - 128	Р - 144	а - 160	· - 176	Ѕ - 192	Ш - 208	р - 224	Ё - 240
Б - 129	С - 145	б - 161	· - 177	Ї - 193	Щ - 209	с - 225	ё - 241
В - 130	Т - 146	в - 162	· - 178	Љ - 194	Ъ - 210	т - 226	ё - 242
Г - 131	У - 147	г - 163	· - 179	Њ - 195	У - 211	у - 227	ё - 243
Д - 132	Ф - 148	д - 164	· - 180	· - 196	Ћ - 212	ф - 228	Ї - 244
Е - 133	Х - 149	е - 165	· - 181	· - 197	Ф - 213	х - 229	ї - 245
Ж - 134	Ц - 150	ж - 166	· - 182	· - 198	Г - 214	ц - 230	Ў - 246
З - 135	Ч - 151	з - 167	· - 183	· - 199	· - 215	ч - 231	ў - 247
И - 136	Ш - 152	и - 168	· - 184	· - 200	· - 216	ш - 232	· - 248
Й - 137	Щ - 153	й - 169	· - 185	· - 201	· - 217	щ - 233	· - 249
К - 138	Ъ - 154	к - 170	· - 186	· - 202	· - 218	ъ - 234	· - 250
Л - 139	Ы - 155	л - 171	· - 187	· - 203	· - 219	ы - 235	· - 251
М - 140	Ь - 156	м - 172	· - 188	· - 204	· - 220	ь - 236	· - 252
Н - 141	Э - 157	н - 173	· - 189	· - 205	· - 221	э - 237	· - 253
О - 142	Ю - 158	о - 174	· - 190	· - 206	· - 222	ю - 238	· - 254
П - 143	Я - 159	п - 175	· - 191	· - 207	· - 223	я - 239	· - 255

Для продолжения нажмите любую клавишу . . .

Краткие итоги

1. Для представления текстовой информации в C++ используются символьные данные и строки.
2. Инициализировать строку можно как *массив символов*.
3. Признаком конца строки является нулевой символ.
4. Обратиться к элементу строки можно по индексу, который соответствует порядковому номеру элемента.
5. Нумерация элементов строки начинается с нуля. Размер строки определяется количеством входящих в нее символов.
6. В C++ предусмотрены различные способы ввода и вывода одиночных символов и строк. При считывании строки с клавиатуры признак конца строки добавляется автоматически.
7. Каждому символу однозначно соответствует его внутренний код.

Для работы со строками используются библиотечные функции, прототипы которых находятся в заголовочных файлах **stdlib.h** и **string.h**.

В программах, в зависимости от типа, вызовы функций для работы со строками задаются в виде:

ИмяФ(СписокАргументов);

или

ИмяПерем=ИмяФ(СписокАргументов);

где **ИмяФ** – имя функции; **СписокАргументов** – список аргументов, передаваемых в тело функции; **ИмяПерем** – идентификатор соответствующего типа.

Например:

y=strlen(st); /*переменной y присвоить значение длины строки st*/

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 бывает необходимо 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').

При вызове функции **strncpy** следует помнить, что, если длина копируемой строки превосходит параметр **kol**, то строка-получатель не будет завершена символом '\0'. В этом случае такой символ надо дописывать в конец строки вручную.

Функции для работы со строками – файл stdlib.h		
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 ($2 \leq baz \leq 36$).
ltoa	char *ltoa (long v, char *str, int baz);	преобразует длинное целое v в строку str . При изображении числа используется основание baz ($2 \leq baz \leq 36$).
ultoa	char *ultoa (unsignedlong v, char *str, int baz);	преобразует беззнаковое длинное целое v в строку str

Функции для работы со строками – файл string.h		
Функция	Прототип	Краткое описание действий
<i>strcat</i>	<i>char *strcat (char *sp, const char *si);</i>	приписывает строку <i>si</i> к строке <i>sp</i> (конкатенация строк)
<i>strchr</i>	<i>char *strchr (const char *str, int c);</i>	ищет в строке <i>str</i> первое вхождение символа <i>c</i>
<i>strcmp</i>	<i>int strcmp (const char *str1, const char *str2);</i>	сравнивает строки <i>str1</i> и <i>str2</i> . Результат отрицателен, если <i>str1<str2</i> ; равен нулю, если <i>str1==str2</i> , и положителен, если <i>str1>str2</i> (сравнение беззнаковое)
<i>strcpy</i>	<i>char *strcpy (char *sp, const char *si);</i>	копирует байты строки <i>si</i> в строку <i>sp</i>
<i>strcspn</i>	<i>int strcspn (const char *str1, const char *str2);</i>	определяет длину первого сегмента строки <i>str1</i> , содержащего символы, не входящие во множество символов строки <i>str2</i>
<i>strdup</i>	<i>char *strdup (const char *str);</i>	выделяет память и переносит в нее копию строки <i>str</i>
<i>strlen</i>	<i>unsigned strlen(const char *str);</i>	вычисляет длину строки <i>str</i>
<i>strlwr</i>	<i>char *strlwr (char *str);</i>	преобразует буквы верхнего регистра в строке в соответствующие буквы нижнего регистра
<i>strncat</i>	<i>char *strncat (char *sp, const char *si, int kol);</i>	приписывает <i>kol</i> символов строки <i>si</i> к строке <i>sp</i> (конкатенация)
<i>strncmp</i>	<i>int strncmp (const char *str1, const char *str2, int kol);</i>	сравнивает части строк <i>str1</i> и <i>str2</i> , причем рассматриваются первые <i>kol</i> символов. Результат отрицателен, если <i>str1<str2</i> ; равен нулю, если <i>str1==str2</i> , и положителен, если <i>str1>str2</i>
<i>strncpy</i>	<i>char *strncpy (char *sp, const char *si, int kol);</i>	копирует <i>kol</i> символов строки <i>si</i> в строку <i>sp</i>
<i>strnicmp</i>	<i>int strnicmp (char *str1, const char *str2, int kol);</i>	сравнивает не более <i>kol</i> символов строки <i>str1</i> и строки <i>str2</i> , не делая различия регистров (см. функцию <i>strncmp</i>)
<i>strnset</i>	<i>char *strnset (char *str, int c, int kol);</i>	заменяет первые <i>kol</i> символов строки <i>str</i> символом <i>c</i>
<i>strpbrk</i>	<i>char *strpbrk (const char *str1, const char *str2);</i>	ищет в строке <i>str1</i> первое появление любого из множества символов, входящих в строку <i>str2</i>
<i>strrchr</i>	<i>char *strrchr (const char *str, int c);</i>	ищет в строке <i>str</i> последнее вхождение символа <i>c</i>

<code>strset</code>	<code>char *strset (char *str, int c);</code>	заполняет строку <code>str</code> заданным символом <code>c</code>
<code>strspn</code>	<code>int strspn (const char *str1, const char *str2);</code>	определяет длину первого сегмента строки <code>str1</code> , содержащего только символы, из множества символов строки <code>str2</code>
<code>strstr</code>	<code>char *strstr (const char *str1, const char *str2);</code>	ищет в строке <code>str1</code> подстроку <code>str2</code> . Возвращает указатель на тот элемент в строке <code>str1</code> , с которого начинается подстрока <code>str2</code>
<code>strtod</code>	<code>double strtod (const char *str, char **endptr);</code>	преобразует <i>символьную константу</i> <code>str</code> в число двойной точности. Если <code>endptr</code> не равен <code>NULL</code> , то <code>*endptr</code> возвращается как указатель на символ, при достижении которого прекращено чтение строки <code>str</code>
<code>strtok</code>	<code>char *strtok (char *str1, const char *str2);</code>	ищет в строке <code>str1</code> <i>лексемы</i> , выделенные символами из второй строки
<code>strtol</code>	<code>long strtol (const char *str, char **endptr, int baz);</code>	Преобразует <i>символьную константу</i> <code>str</code> к значению "длинное число" с основанием <code>baz</code> ($2 \leq baz \leq 36$). Если <code>endptr</code> не равен <code>NULL</code> , то <code>*endptr</code> возвращается как указатель на символ, при достижении которого прекращено чтение строки <code>str</code>
<code>strupr</code>	<code>char *strupr (char *str);</code>	преобразует буквы нижнего регистра в строке <code>str</code> в буквы верхнего регистра

Сравнение строк с помощью функции ***strcmp*** производится побайтово в лексикографическом порядке, то есть в порядке прохождения соответствующих байтов строк в таблице кодирования. Именно поэтому значение элементов в строках зависит от регистра.

При использовании библиотечных функций следует иметь в виду, что указатель на строку и имя массива символов указывают адрес размещения строки в памяти. Это означает, что изменения значений элементов строки сохраняются после завершения работы функции. Чтобы не допустить изменений в строке, используется указатель на константу, которая не позволит модифицировать данные, хранящиеся по адресуемой области памяти.

Пример 74. Программа демонстрирует работу функций из файла `stdlib.h`

Visual Studio 2010

```
#include<iostream>
#include <windows.h>
#include <stdlib.h>
int main()
{
    SetConsoleOutputCP(1251);
```

```

char sv[]="23.547",
      si[]="1234",
      sl[]="-112424",
      st1[15], st2[25], st3[15];
long l,t=457821;
l=atol(sl);
printf("Преобразование строки в длинное целое число = %ld\n", l);
printf("Преобразование строки в вещественное число = %f\n", atof(sv));
printf("Преобразование строки в целое число = %d\n", atoi(si));
ultoa(t,st1,10);
printf("Преобразование длинного целого числа в строку = %s\n", st1);
printf("Преобразование длинного целого числа в строку = %s\n", ultoa(t,st2,2));
printf("Преобразование длинного целого числа в строку = %s\n", ultoa(t,st3,16));
return 0;
}

```

Результат выполнения программы:

```

Преобразование строки в длинное целое число = -112424
Преобразование строки в вещественное число = 23.547000
Преобразование строки в целое число = 1234
Преобразование длинного целого числа в строку = 457821
Преобразование длинного целого числа в строку = 1101111110001011101
Преобразование длинного целого числа в строку = 6fc5d
Для продолжения нажмите любую клавишу . . .

```

Visual Studio 2019

```

#include<iostream>
#include <windows.h>
#include <stdlib.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    char sv[] = "23.547",
          si[] = "1234",
          sl[] = "-112424",
          st1[15], st2[25], st3[15];
    long l, t = 457821;
    l = atol(sl);
    printf("Преобразование строки в длинное целое число = %ld\n", l);
    printf("Преобразование строки в вещественное число = %f\n", atof(sv));
    printf("Преобразование строки в целое число = %d\n", atoi(si));
    _ultoa_s(t, st1, 10);
    printf("Преобразование длинного целого числа в строку = %s\n", st1);
    _ultoa_s(t, st2, 2);
    printf("Преобразование длинного целого числа в строку = %s\n", st2);
    _ultoa_s(t, st3, 16);
    printf("Преобразование длинного целого числа в строку = %s\n", st3);
    return 0;
}

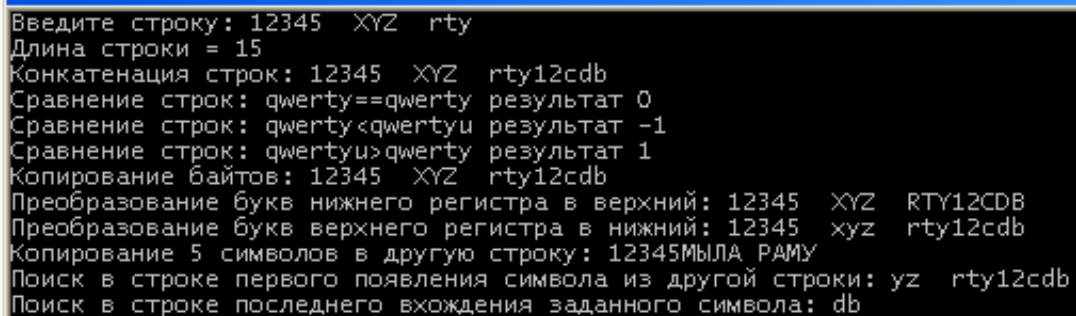
```

Пример 75. Программа демонстрирует работу функций из файла string.h

Visual Studio 2010

```
int main()
{ SetConsoleOutputCP(1251);
  char st[50],sp[100], str[20]="МАМА МЫЛА РАМУ",
    si[]="qwerty",
    sl[]="qwerty",
    sw[]="qwertyu";
  int len=0, sravn1, kol=5;
  printf("Введите строку: ");
  gets(st);
  len=strlen(st);
  printf("Длина строки = %d\n", len);
  printf("Конкатенация строк: %s\n", strcat(st,"12cdb"));
  sravn1=strcmp(si,sl);
  printf("Сравнение строк: %s==%s результат %d\n", si,sl,sravn1);
  printf("Сравнение строк: %s<%s результат %d\n", si,sw,strcmp(si,sw));
  printf("Сравнение строк: %s>%s результат %d\n", sw,si,strcmp(sw,si));
  printf("Копирование байтов: %s\n", strcpy(sp,st));
  printf("Преобразование букв нижнего регистра в верхний: %s\n",strupr(st));
  printf("Преобразование букв верхнего регистра в нижний: %s\n",strlwr(st));
  printf("Копирование %d символов в другую строку: %s\n", kol, strncpy(str,st,kol));
  printf("Поиск в строке первого появления символа из другой строки: %s\n",
  strpbrk(st,si));
  printf("Поиск в строке последнего вхождения заданного символа: %s\n",
  strrchr(st,'d'));
  return 0;
}
```

Результат выполнения программы:



```
Введите строку: 12345 XYZ rty
Длина строки = 15
Конкатенация строк: 12345 XYZ rty12cdb
Сравнение строк: qwerty==qwerty результат 0
Сравнение строк: qwerty<qwertyu результат -1
Сравнение строк: qwertyu>qwerty результат 1
Копирование байтов: 12345 XYZ rty12cdb
Преобразование букв нижнего регистра в верхний: 12345 XYZ RTY12CDB
Преобразование букв верхнего регистра в нижний: 12345 xyz rty12cdb
Копирование 5 символов в другую строку: 12345МЫЛА РАМУ
Поиск в строке первого появления символа из другой строки: yz rty12cdb
Поиск в строке последнего вхождения заданного символа: db
```

Visual Studio 2019

```
#include <iostream>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
using namespace std;
int main()
{
  SetConsoleOutputCP(1251);
  char st[50]="", sp[100] = "", str[20] = "МАМА МЫЛА РАМУ",
    si[] = "qwerty",
    sl[] = "qwerty",
```

```

    sw[] = "qWerty";
    int len = 0, sravn1, kol = 5;
    printf("Введите строку: ");
    cin.getline(st, 50, '\n');
    len = strlen(st);
    printf ("Длина строки = %d\n", len);
    strcat_s(st, "12cdb");
    printf ("Конкатенация строк: %s\n", st);
    sravn1 = strcmp(si, sl);
    printf ("Сравнение строк: %s==%s результат %d\n", si, sl, sravn1);
    printf ("Сравнение строк: %s<%s результат %d\n", sw, si, strcmp(sw,si));
    printf ("Сравнение строк: %s>%s результат %d\n", si, sw, strcmp(si,sw));
    strcpy_s(sp, st);
    printf("Копирование байтов: %s\n", sp);
    _strupr_s(st);
    printf ("Преобразование букв нижнего регистра в верхний: %s\n", st);
    _strlwr_s(st);
    printf ("Преобразование букв верхнего регистра в нижний: %s\n", st);
    strncpy_s(str, st, kol);
    printf ("Копирование %d символов в другую строку: %s\n", kol, str);
    printf ("Поиск в строке первого появления символа из другой строки: %s\n", strpbrk(st,
si));
    printf ("Поиск в строке последнего вхождения заданного символа: %s\n", strrchr(st, 'd'));
    return 0;
}

```

```

Введите строку: 12345 XYZ rty
Длина строки = 14
Конкатенация строк: 12345 XYZ rty12cdb
Сравнение строк: qwerty==qwerty результат 0
Сравнение строк: qwerty<qwerty результат -1
Сравнение строк: qwerty>qwerty результат 1
Копирование байтов: 12345 XYZ rty12cdb
Преобразование букв нижнего регистра в верхний: 12345 XYZ RTY12CDB
Преобразование букв верхнего регистра в нижний: 12345 xyz rty12cdb
Копирование 5 символов в другую строку: 12345
Поиск в строке первого появления символа из другой строки: yz rty12cdb
Поиск в строке последнего вхождения заданного символа: db

```

Из файла <ctype.h>:

int isdigit(int); // определяет, цифра или нет

int isalpha(int); //буква

int isupper(int); //буква в верхнем регистре

int islower(int); //буква в нижнем регистре

int isspace(int); //символ – разделитель

int ispunct(int); //символ пунктуации (ни один из вышеупомянутых)

int isalnum(int); //буква или цифра

int toupper(int); //перевод в верхний регистр

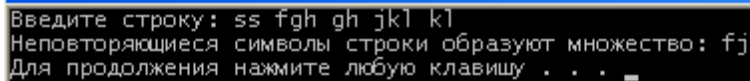
int tolower(int); //перевод в нижний регистр

Пример 75. Поиск множества неповторяющихся символов строки

Visual Studio 2010

```
int main()
{
    SetConsoleOutputCP(1251);
    char st[80];
    int i,j,flag,len;
    printf("Введите строку: ");
    gets(st);
    len=strlen(st);          //длина строки
    printf("Неповторяющиеся символы строки образуют множество: ");
    for (i=1; i<len; i++)
    {
        flag=0;              //флаг проверки на совпадение
        for (j=0; j<i; j++)    //сравнение символа с предыдущими
            if (st[i]==st[j]) flag=1;
        for (j=i+1; j<len; j++) //сравнение символа с последующими
            if (st[i]==st[j]) flag=1;
        if (flag==0) printf("%c", st[i]);
    }
    cout<<"\n";
    return 0;
}
```

Результат выполнения программы:



```
Введите строку: ss fgh gh jkl kl
Неповторяющиеся символы строки образуют множество: fj
Для продолжения нажмите любую клавишу . . . _
```

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

1. Для работы со строками в языке C++ предусмотрены стандартные функции, прототипы которых включены в *стандартные библиотеки **stdlib.h** и **string.h***.
2. При *обращении к функциям* для работы со строками следует учитывать, что изменение значений элементов строк сохраняются после завершения работы функции.
3. Перед использованием строки в программном коде ее необходимо проинициализировать. Неинициализированные строки могут привести к некорректной работе программы.
4. В некоторых стандартных функциях по работе со строками следует проводить контроль длин параметров.

5. Результат работы некоторых функций требует принудительного добавления к строке *символа конца строки*.
6. Значения элементов строк зависят от регистра.
7. Изменение регистра символов кириллицы в программе может выполняться некорректно.

АЛГОРИТМ ВЫДЕЛЕНИЯ СЛОВ

Часто при обработке строк требуется выделять слова. Словом будем считать последовательность любых символов, отличных от пробелов. Задача выделения слов является важной подзадачей в лексическом анализе текста программы. В этом случае слова называются лексемами и определяются как минимальные единицы языка, имеющие самостоятельный смысл.

В качестве примера рассмотрим строку *str*, в которой слова разделяются пробелами.

Если исходная строка не должна изменяться, то пробелы не удаляют, а пропускают.

Цикл для пропуска пробелов между словами:

```
while (str[i] == ' ' && i < len)
    ++i;           //пропустить пробелы
```

Слова можно пропустить аналогичным циклом:

```
while (str[i] != ' ' && i < len)
    ++i;           // пропустить все символы слова
```

Эти два цикла должны быть включены во внешний цикл, который закончится тогда, когда закончится строка.

Пример 9.1. Вывод на экран всех слов строки *str* (Алгоритм 1):

```
int main()
{
    int i = 0, begin = 0, end = 0;
    char str[100], sl[100];
    cout << "InputLine: \n";
    cin.getline(str, 100, '\n');
    int len = strlen(str);
    sl[0] = '\0';
    cout << "Slova: \n ";
    while (i < len)           //цикл прохода по строке
    {
        while (str[i] == ' ' && i < len)
            ++i;           //пропустить пробелы
        begin = i;           // номер первого символа слова
        while (str[i] != ' ' && i < len)
            ++i;           // пропустить все символы слова
        end = i;           // номер символа, следующего за последним символом слова
        strncpy_s(sl, &str[begin], end - begin); //записать слово в массив
        sl[end - begin] = '\0'; // добавить символ конца строки
        cout << sl << '\n'; ; // вывод слова
    }
    return 0;
}
```

```

InputLine:
C++ один из популярных языков программирования
Slova:
C++
один
из
популярных
языков
программирования

```

Пример 9.2. Вывод на экран всех слов str (алгоритм 2).

```

int main()
{
    char str[100], sl[100];
    int k = 0, i;
    cout << "InputLine: \n";
    cin.getline(str, 100, '\n');
    strcat_s(str, " ");
    cout << "Slova:\n ";
    sl[0] = '\0';
    for (i = 0; i < strlen(str); i++)           //проход по строке
        if (str[i] != ' ')                     //если символ строки отличный от пробела
            sl[k++] = str[i]; //записываем его в переменную sl для хранения слова
        else                                   //если символ строки пробел
        {
            if (strlen(sl) > 0 )               //если длина слова отличная от нуля
                {sl[k] = '\0'; cout << sl << '\n'; } //добавить '\0' и вывести слово
            sl[0] = '\0';
            k = 0;
        }
    return 0;
}

```

```

InputLine:
C++ один из популярных языков программирования
Slova:
C++
один
из
популярных
языков
программирования

```

Строки и указатели

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

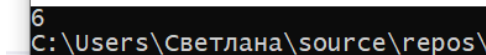
Поскольку строка всегда заканчивается нулевым символом, можно увеличивать указатель на 1, пока очередным символом не станет ноль. Например:

```
while ( *st++ ) { ... }
```

st разыменовывается, и получившееся значение проверяется на истинность. Любое отличное от нуля значение считается истинным, и, следовательно, цикл заканчивается, когда будет достигнут символ с кодом 0. Операция инкремента ++ прибавляет 1 к указателю st и таким образом сдвигает его к следующему символу. Поскольку указатель может содержать нулевое значение (ни на что не указывать), перед

операцией разыменования его следует проверять. Вот как может выглядеть вычисление длины строки.

```
void main()
{
    const char *st= "123456";
    int cnt = 0;
    if ( st )
        while ( *st++ )
            ++cnt;
    cout<<cnt;
}
```




Строка встроенного типа может считаться пустой в двух случаях: если указатель на строку имеет нулевое значение (тогда у нас вообще нет никакой строки) или указывает на массив, состоящий из одного нулевого символа (то есть на строку, не содержащую ни одного значимого символа).

```
char *pc1 = 0;           // pc1 не адресует никакого массива символов
const char *pc2 = "";    // pc2 адресует нулевой символ
```

При разработке функций для работы со строками в большинстве случаев целесообразно применять указатели. Приведем примеры фрагментов программ:

*/*Пример пользовательской функции копирования строки s2 в s1*/*

```
void main()
{
    char s1[] = "123456"; char s2[] = "789";
    char *p1 = s1, *p2 = s2; //указатели инициализированы на начало строк
    while ((*p1++ = *p2++) != 0);
    // вывод строки s1
    p1 = s1; //указатель инициализирован на начало строки s1
    while ( *p1 != '\0') {cout<<*p1; p1++;}
}
```



Следующий пример демонстрирует, что использование нулевого ограничителя упрощает различные операции над строками.

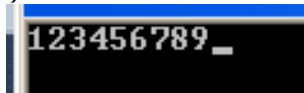
*/*Пример пользовательской функции конкатенации*/*

```
void main()
{
    char s1[] = "123456"; char s2[] = "789";
    char *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);
    // вывод строки s1
}
```

```

p1 = s1;      //указатель инициализирован на начало строки s1
while ( *p1 != '\0') {cout<<*p1; p1++;}
}

```



Пример 1.

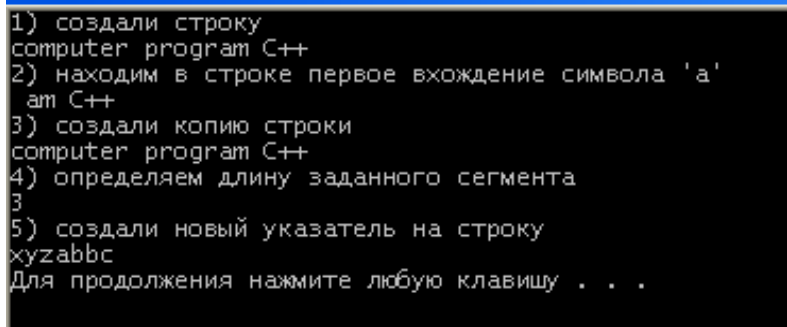
/*Демонстрация работы с указателями и с функциями для обработки строк*/

Visual Studio 2010

```

#include <iostream>
#include <windows.h>
#include <string.h>
using namespace std;
int main()
{SetConsoleOutputCP(1251);
char string[100], temp[100], *result, simvol;
int numresult, res;
/*создает строку "computer program C++ " посредством
использования strcpy и strcat*/
strcpy(string, "computer");
result = strcat(string, " program C++");
printf("1) создали строку\n%s\n", result);
/*находит строку, в которой первый раз обнаружено 'a'*/
simvol='a';
result = strchr(string,simvol);
printf("2) находим в строке первое вхождение символа \'%c\' \n %s\n",simvol,result);
/* создает копию строки */
result = strcpy(temp,string);
printf("3) создали копию строки\n%s\n",result);
/* находит "a","b","c" в строке */
strcpy(string,"xyzabbc");
res = strcspn(string,"abc"); /* определяет длину первого сегмента строки string,
содержащего символы, не входящие во множество символов строки "abc"*/
printf("4) определяем длину заданного сегмента \n%d\n",res);
/*создает новый указатель на строку для дублирования
строки*/
result = strdup(string); /*выделяет память и переносит в нее копию строки string*/
printf("5) создали новый указатель на строку \n%s\n",result);
return 0;
}

```



Visual Studio 2019

```

#include <iostream>
#include <windows.h>

```

```

#include <string.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    char string[100], temp[100], *result=0, simvol;
    int numresult, res;
    /*создает строку "computer program C++ " посредством
    использования strcpy и strcat*/
    strcpy_s(string, "computer");
    strcat_s(string, " program C++");
    result = string;
    printf("1) создали строку\n%s\n", result);
    /*находит строку, в которой первый раз обнаружено 'a'*/
    simvol = 'a';
    result = strchr(string, simvol);
    printf("2) находим в строке первое вхождение символа '%c'\n %s\n", simvol, result);
    /* создает копию строки */
    strcpy_s(temp, string);
    result = temp;
    printf("3) создали копию строки\n%s\n", result);
    /* находит "a","b","c" в строке */
    strcpy_s(string, "xyzabbc");
    res = strcspn(string, "abc"); /* определяет длину первого сегмента строки string,
    содержащего символы, не входящие во множество символов строки "abc"*/
    printf("4) определяем длину заданного сегмента \n%d\n", res);
    /*создает новый указатель на строку для дублирования
    строки*/
    result = _strdup(string); /*выделяет память и переносит в нее копию строки string*/
    printf("5) создали новый указатель на строку \n%s\n", result);
    return 0;
}

```

В предыдущих примерах рассматривалось присвоение указателю адреса только первого элемента символьного массива. Однако это можно делать и с адресом любого отдельного элемента массива путем добавления символа '&' к индексированному имени. Особенно удобно пользоваться этим правилом при выделении подстроки.

Например, программа выводит на экран часть введенной строки после первого пробела:

Пример 2.

*/*Вывести на экран часть строки после первого пробела*/*

Visual Studio 2010

```

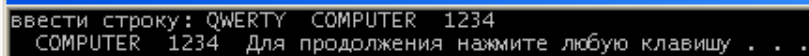
#include <windows.h>
using namespace std;
int main()
{SetConsoleOutputCP(1251);
    char s[80], *p;
    int i;
    printf("ввести строку: ");
    gets(s);
    /*найти первый пробел или конец строки*/
}

```

```

for(i=0; s[i] && s[i]!=' '; i++);
p = &s[i];
printf(p);
return 0;
}

```



```

ВВЕСТИ СТРОКУ: QWERTY COMPUTER 1234
COMPUTER 1234 Для продолжения нажмите любую клавишу . . .

```

Visual Studio 2019

```

#include <iostream>
#include <windows.h>
#include <string.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    char s[80], * p;
    int i;
    printf("ВВЕСТИ СТРОКУ: ");
    cin.getline(s, 50, '\n');
    /*найти первый пробел или конец строки*/
    for (i = 0; s[i] && s[i] != ' '; i++);
    p = &s[i];
    printf(p);
    return 0;
}

```

В этой программе **p** будет указывать либо на *пробел*, если он есть, либо на ноль, если в строке нет пробелов. Если **p** указывает на *пробел*, то *программа* выведет на экран его и затем *остаток* строки. Если **p** укажет на ноль, то на экран ничего не выводится.

Пример 3:

//Выводит каждое отдельное слово и подсчитывает его длину

```

#include <iostream>
#include <windows.h>
using namespace std;
int main()
{
    SetConsoleOutputCP(1251);
    char text[100],*p, *razd=" .,";
    int dlina;
    puts ("Введите текст ");
    gets(text);
    p=strtok(text,razd); // Выделение первого слова текста
    while (p) // Пока можно выделить слово
    {
        dlina=strlen(p); // Определение длины слова
        cout << "\n слово " << p << " длина = " << dlina << "\n";
        p=strtok(NULL,razd); //Выделение второго, третьего, и т.д. слов
    }
    return 0;
}

```

```

Введите текст
12345.1234 56,qwerty*.

слово 12345 длина = 5
слово 1234 длина = 4
слово 56 длина = 2
слово qwerty* длина = 7

```

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

- При передаче строки как *параметра функции* не указывается *длина*, так как ограничителем является *символ конца строки*.
- Строки передаются в функции в качестве параметров как массивы символов или как указатели типа **char**.
- 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.
- 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**;
- 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*.
- 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

```

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.

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.